НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютеризованих систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Хмарне архітектурне рішення сервісу обробки ліцензій продуктів та послуг

Виконав студент IV курсу, групи ІТ-84в
(шифр групи)
Осадчий Мирослав Віталійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Ліщук К. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації ст. вик., Вітковська І. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ІСТ, к.т.н., Писаренко А. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Осадчому Мирославу Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Хмарне архітектурне рішення сервісу обробки ліцензій продуктів та послуг

керівник проєкту Ліщук Катерина Ігорівна, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «02» червня 2023 р. №2160-с

2. Термін подання студентом проєкту «17» червня 2023 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки

1) Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі.
2) Аналіз вимог до програмного забезпечення: розробка функціональних та нефункціональних вимог
3) Моделювання та конструювання програмного забезпечення, моделювання та аналіз програмного забезпечення, архітектура програмного забезпечення, конструювання програмного забезпечення, аналіз безпеки даних
4) Аналіз якості та тестування програмного забезпечення, аналіз якості програмного забезпечення, опис процесів тестування, опис контрольного прикладу
5) Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань.

2) Схема структурна діаграми послідовностей.

3) Схема структурна класів програмного забезпечення.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» березня 2023 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	05.02.2023	
2	Аналіз існуючих методів розв’язання задачі	12.02.2023	
3	Постановка та формалізація задачі	23.02.2023	
4	Розробка інформаційного забезпечення	02.03.2023	
5	Алгоритмізація задачі	13.03.2023	
6	Обґрунтування вибору використаних технічних засобів	18.03.2023	
7	Розробка програмного забезпечення	05.04.2023	
8	Налагодження програми	03.05.2023	
9	Виконання графічних документів	11.05.2023	
10	Оформлення пояснювальної записки	17.05.2023	
11	Подання ДП на попередній захист	02.06.2023	
12	Подання ДП рецензенту	12.06.2023	
13	Подання ДП на основний захист	17.06.2023	

Студент _____ Мирослав ОСАДЧИЙ

(підпис) (ініціали, прізвище)

Керівник _____ Катерина ЛІЩУК

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 50 таблиць, 9 рисунків та 7 джерел – загалом 65 сторінки.

Мета роботи – створення уніфікованого шаблону хмарного сервісу, який буде відповідати за взаємодію між користувачами та ліцензіями, між власником та його продуктом, між споживачем та продуктом, таким чином мінімізувати витрати бізнесу на власну розробку і дати змогу сфокусуватися на бізнес деталях. Вирішити проблему масштабованості, відмовостійкості та вартості розробки і підтримки такого застосунку, використання на всіх сучасних операційних системах. Створення відкритого програмного інтерфейсу що забезпечує можливість інтеграції з сервісом додатків інших розробників.

Об'єкт дослідження – уніфікований шаблон сервісу для взаємодії між користувачем та ліцензіями продуктів чи послуг.

Предмет дослідження: сучасні підходи розробки хмарних додатків, реалізація архітектурних патернів під час роботи з хмарними сервісами.

У першому розділи наведені загальні положення, опис та аналіз предметної області, сформульовані функціональні і нефункціональні вимоги, головна мета, цілі та призначення програми.

У другому розділі описано моделювання та бізнес-процеси програмного забезпечення, розроблена архітектура та компоненти. Наведено детальний опис модулів, класів та функцій.

У третьому розділі проаналізовано якість програмного забезпечення, описано процес тестування та наведені приклади тестів.

У четвертому розділі наведено опис розгортання та роботи програми.

Результатом роботи є розробка інтегрованого модуля з обробки ліцензій продуктів та послуг.

КЛЮЧОВІ СЛОВА: .NET, IAC, AWS, SDK, ХМАРНА АРХІТЕКТУРА, API, IDE.

ABSTRACT

The explanatory note of the diploma project consists of four sections and includes 50 tables, 9 figures, and 7 references, totaling 65 pages.

The aim of the project is to create a unified template for a cloud service that will be responsible for the interaction between users and licenses, owners and their products, consumers and the product, thus minimizing business costs for in-house development and enabling focus on business details. It aims to address scalability, fault tolerance, and the cost of developing and supporting such an application, for use on all modern operating systems. The creation of an open application programming interface (API) will provide the ability to integrate with the services of other developers.

The object of the research is a unified service template for the interaction between users and product or service licenses.

The subject of the research is modern approaches to developing cloud applications and implementing architectural patterns when working with cloud services.

The first section presents general provisions, a description and analysis of the subject area, formulation of functional and non-functional requirements, the main goal, objectives, and purpose of the software.

The second section describes the modeling and business processes of the software, architecture and components. It provides a detailed description of modules, classes, and functions.

The third section analyzes the quality of the software, describes the testing process, and provides examples of tests.

The fourth section provides a description of the deployment and operation of the software.

The result of the work is the development of an integrated module for processing product and service licenses.

KEYWORDS: .NET, IAC, AWS, SDK, CLOUD ARCHITECTURE, API, IDE.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4	КПІ.ІТ-84в23.045440.00.91	Завдання на дипломний проєкт	4	
2	А4	КПІ.ІТ-84в23.045440.01.91	Технічне завдання	13	
3	А4	КПІ.ІТ-84в23.045440.02.81	Пояснювальна записка	65	
4	А4	КПІ.ІТ-84в23.045440.03.12	Текст програми	123	
5	А4	КПІ.ІТ-84в23.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІТ-84в23.045440.05.34	Керівництво користувача	5	
7	А3	КПІ.ІТ-84в23.045440.06.99	Схема структурна варіантів	1	
			використань		
8	А3	КПІ.ІТ-84в23.045440.07.99	Схема структурна	1	
			послідовностей системи		
9	А3	КПІ.ІТ-84в23.045440.08.99	Схема структурна класів	3	
			програмного забезпечення		

					КПІ.ІТ-84в23.045440.00.90		
Змін.	Арк.	№ докум.	Підп.	Дата			
Розроб.		Осадчий М. В.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Ліщук К. І.					
Н.контр.		Ліщук К.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-84в	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2023 р.

ХМАРНЕ АРХІТЕКТУРНЕ РІШЕННЯ СЕРВІСУ ОБРОБКИ ЛІЦЕНЗІЙ ПРОДУКТІВ ТА ПОСЛУГ

Технічне завдання

КПІ.ІТ-84в23.045440.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Катерина ЛІЩУК

Нормоконтроль:

_____ Ірина ВІТКОВСЬКА

Виконавець:

_____ Мирослав ОСАДЧИЙ

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: хмарне архітектурне рішення сервісу обробки ліцензій продуктів та послуг.

Галузь застосування: інтегрований модуль для функціонування бізнес застосунків.

Наведене технічне завдання поширюється на розробку хмарного архітектурного рішення 045440, яке використовується для маніпулювання сутностями ліцензій на продукти чи послуги, інтегрування у сучасні бізнес застосунки.

.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки хмарного архітектурного рішення сервісу обробки ліцензій продуктів та послуг є завдання на дипломне проектування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для автоматизації роботи з ліцензіями.

Метою розробки є створення уніфікованого шаблону хмарного сервісу, який буде відповідати за взаємодію між користувачами та ліцензіями, між власником та його продуктом, між споживачем та продуктом, таким чином мінімізувати витрати бізнесу на власну розробку і дати змогу сфокусуватися на бізнес деталях. Вирішити проблему масштабованості, відмовостійкості та вартості розробки і підтримки такого застосунку, використання на всіх сучасних операційних системах. Створення відкритого програмного інтерфейсу що забезпечує можливість інтеграції з сервісом додатків інших розробників.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

Програмним забезпеченням не передбачено наявність користувацького інтерфейсу.

4.1.2 Для користувача

– створення, видалення, оновлення та перегляд продуктів;

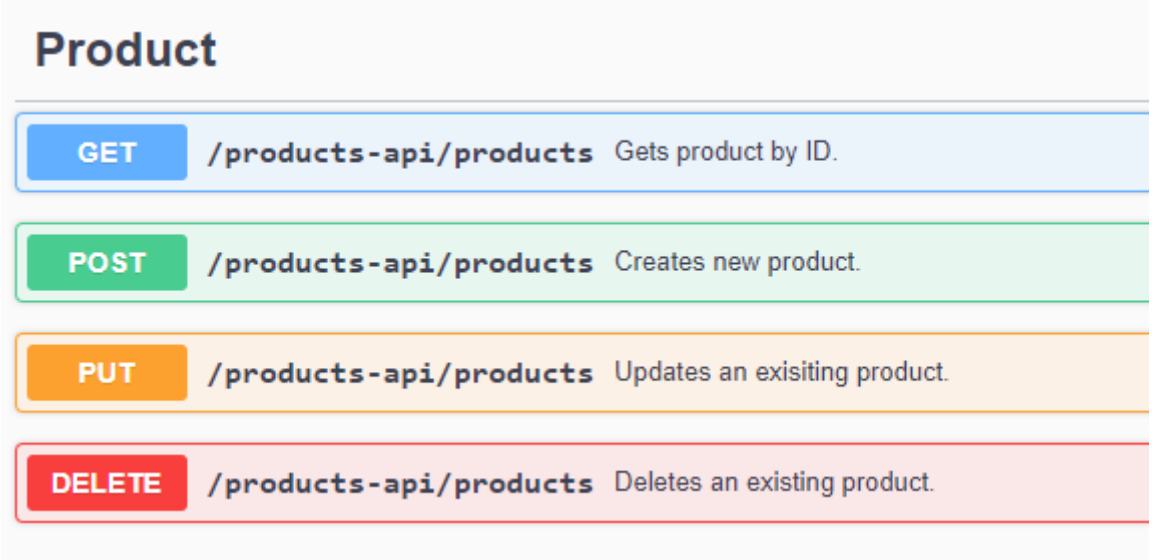


Рисунок 4.1 – API продуктів

– створення, видалення, оновлення та перегляд користувачів;

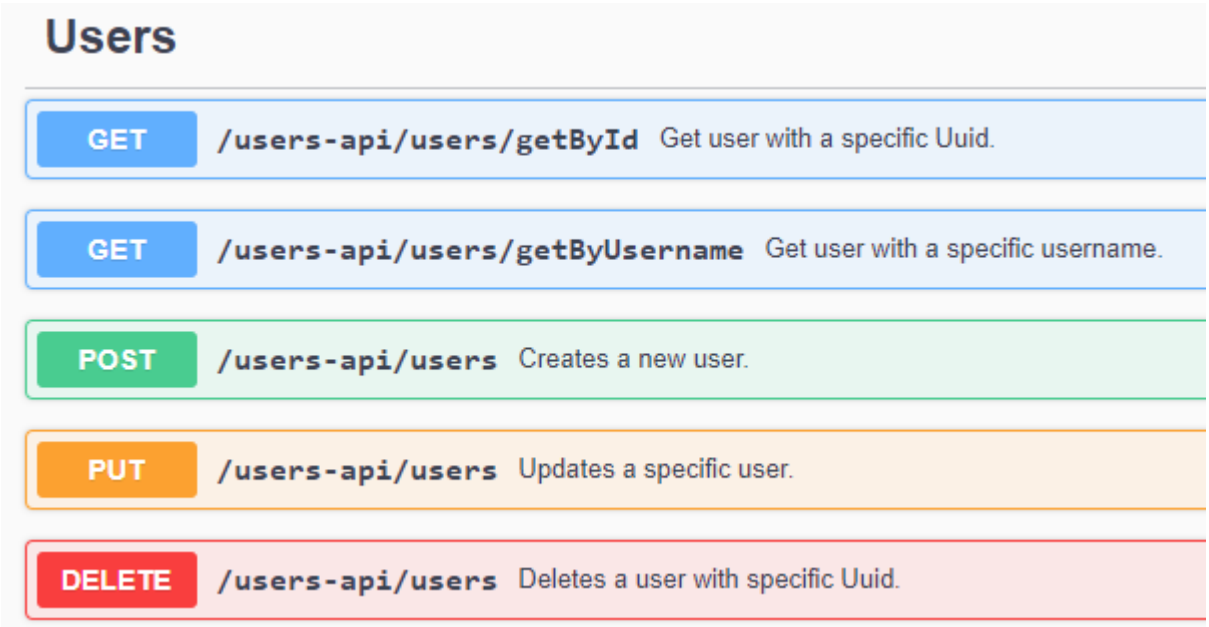


Рисунок 4.2 – API користувачів

– створення, видалення, оновлення та перегляд ліцензій;

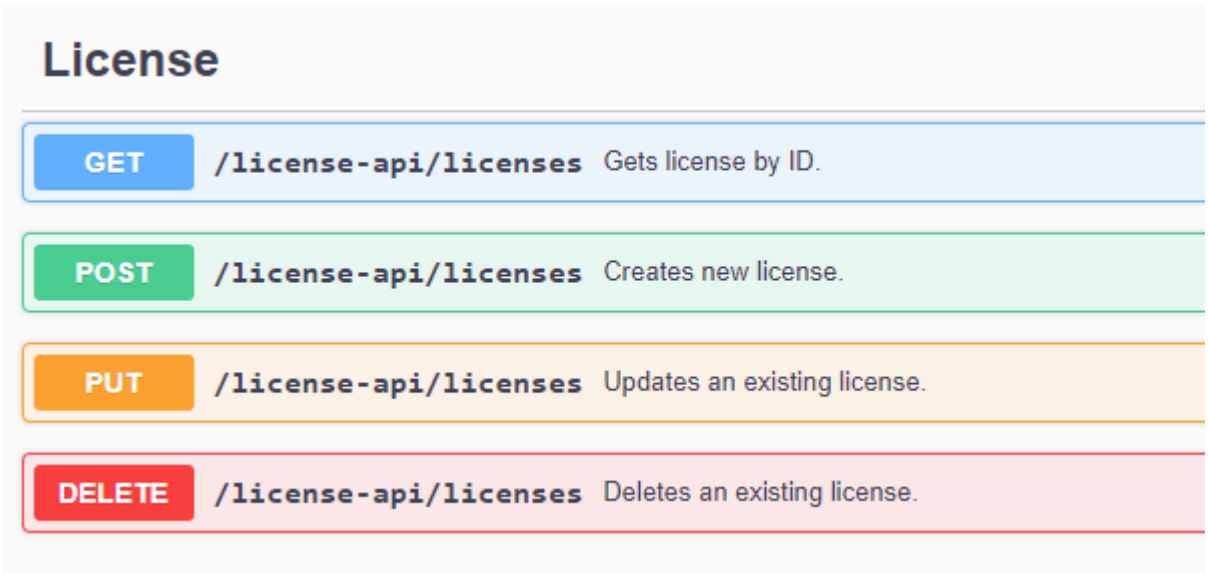


Рисунок 4.3 – API ліцензій

– створення, видалення та перегляд прав на продукти;

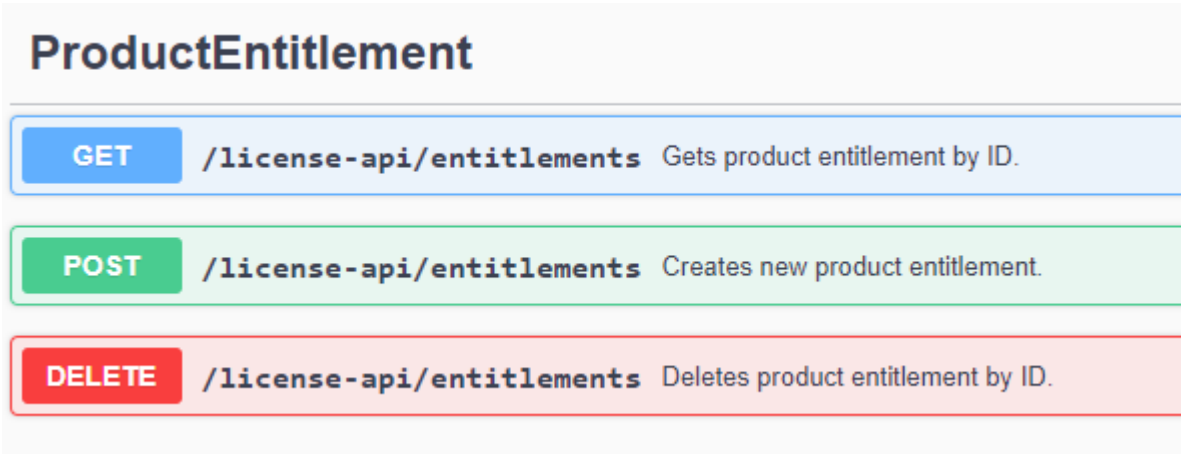


Рисунок 4.4 – API прав на продукти

4.1.3 Для адміністратора системи

Системою не передбачено наявність ролі адміністратора.

4.1.4 Додаткові вимоги

Інтеграція користувача зі сторонніми сервісами.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Необхідний доступ до мережі інтернет, задля функціонування на хмарному обчислювальному сервісі AWS Lambda.

Клієнтська частина програмного забезпечення повинна функціонувати у веб-браузері на ПК.

Мінімальна конфігурація технічних засобів:

– об‘єм ОЗП: 1 ГБ;
– підключення до мережі Інтернет зі швидкістю від 10 мегабіт.

Рекомендована конфігурація технічних засобів:

– об‘єм ОЗП: 16 Гб;
– підключення до мережі Інтернет зі швидкістю від 100 мегабіт.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати на будь-яких операційних системах з підтримкою веб-браузерів Google Chrome, Microsoft Edge, Mozilla Firefox, Opera.

4.5.1 Вимоги до вхідних даних

Вимоги до вхідних даних відсутні.

4.5.2 Вимоги до вихідних даних

Вимоги до вихідних даних відсутні.

4.5.3 Вимоги до мови розробки

Розробка виконана на мові програмування C# (.NET Core 6)

4.5.4 Вимоги середовища розробки

Visual Studio 2022 (+AWS SDK), Visual Studio Code.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді текстових файлів з розширенням .cs.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні вимоги не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;

– технічне завдання;

– текст програми;

– програма та методика тестування;

– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;

– схема структурна послідовностей системи;

– схема структурна класів програмного забезпечення.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	05.02.2023	
2.	Розробка технічного завдання	12.02.2023	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	15.03.2023	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	18.03.2023	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	29.04.2023	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	03.05.2023	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	05.05.2023	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	11.05.2023	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	17.05.2023	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка

до дипломного проєкту

на тему: Хмарне архітектурне рішення сервісу обробки ліцензій продуктів та послуг

КПІ.ІТ-84в23.045440.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE – Integrated Development Environment – інтегроване середовище розробки.

VS – Visual Studio.

VS Code – Visual Studio Code.

AWS – Amazon Web Services.

API – Application programming interface, прикладний програмний Інтерфейс.

SDK – Software development kit.

БД – База даних.

UC – use cases – варіанти використання.

IaC – infrastructure as a code – інфраструктура, як код.

ВСТУП

У сучасному світі, де технології розвиваються та інтегруються у наше життя дуже швидко, важко уявити життєдіяльність сучасного бізнесу без їх активного використання. Конкуренція вимагає наявність швидких та надійних рішень з розробки та розгортки бізнес-застосувань. Боротьба за клієнта, економія коштів та оптимізація усіх можливих процесів своєї сфери також вимагають особливого підходу до розробки програмного забезпечення. Як приклад основного застосування хмарних застосунків для обробки клієнтів та наявних послуг, можна привести банківську систему. Основною сферою виступає робота з фінансами, але не малозначним є саме збереження та взаємодія великої кількості даних між клієнтами та сервісом. Для користувача важливо мати змогу легко відслідковувати свої можливості та працювати зі своїми придбаними продуктами та наявними послугами. У такому випадку виникає проблема чіткого розподілу прав та обов'язків користувачів у системі. Потрібно керувати відразу декількома процесами паралельно або вносити корективи через різні фактори або ситуації. Виходячи з цього, сучасна тенденція вимагає оптимізації усіх можливих процесів. Існує потреба у створенні уніфікованого шаблону для подібних застосунків, який може виконувати певні потреби набагато краще ніж інші варіанти на ринку, полегшить та прискорить розробку, не зважаючи на масштаб проекту чи безпосередню сферу застосувань.

Для розробки подібного ПЗ необхідно мати певну кількість досвідчених розробників та бізнес-аналітиків, які враховують всі технологічні та законодавчі обмеження. Основна проблема полягає в тому, що механізм взаємодії між різними типами клієнтів та доступними їм послугами необхідно реалізовувати кожного разу майже однаково. У результаті цього постає проблема створення уніфікованого шаблону, який можна застосувати для будь якого додатку, та який буде легко масштабуватись у будь яку сторону.

Мета даного дипломного проекту – створення уніфікованого шаблону хмарного сервісу, який буде відповідати за взаємодію між користувачами та ліцензіями, між власником та його продуктом, між споживачем та продуктом, таким чином мінімізувати витрати бізнесу на власну розробку і дати змогу сфокусуватися на бізнес деталях. Вирішити проблему масштабованості такого застосунку та використання на всіх сучасних операційних системах.

Вважаю дану тему дипломного проекту доцільною та перспективною для розвитку якості програмного забезпечення та простоти його підтримки.

Об'єкт дослідження – уніфікований шаблон сервісу для взаємодії між користувачем та ліцензіями продуктів чи послуг.

Предмет роботи – методи масштабування ПЗ, теорія бізнес рішень у взаємодії між користувачем та сервісом, використання хмарних сервісів у сучасній розробці.

Метою роботи є створення хмарного сервісу для ефективної роботи бізнесу та комфортної роботою зі сторони команди, яка його буде підтримувати.

Практичне значення ПЗ полягає у реалізації продукту, який можна легко інтегрувати у будь який існуючий бізнес застосунок, має змогу приймати велику кількість запитів від користувачів з мінімальною кількістю помилок під час навантаження; буде гнучким і може бути легко масштабованим або зміненим згідно з корпоративними вимогами.

Дипломний проект складається з кількох розділів: вступ, основні розділи, висновки та список використаних джерел.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальні положення

Предметна область визначається реалізацією програмного інтерфейсу інтегрованого модулю з обробки ліцензій.

Створення, обробка та використання ліцензій – це ключовий елемент сучасного бізнес застосунку. Надважливо чітко розділяти кількість та якість пропонованих послуг між групами користувачів. Існує велика кількість можливих реалізацій модуля для обробки ліцензій, але для такого поширеного програмного забезпечення, йому бракує уніфікації. Сучасна індустрія безпосередньо залучена у створенні нових сервісів під наглядом великих провідних компаній, які надають низку послуг у певних сферах комерційної діяльності. Будуються вони на використанні переліку дуже різних технологій, але основним на даний момент є шаблон база даних + веб інтерфейс.

Використання ж мікро сервісів створює більше можливостей для автоматизації та управління, підвищує мобільність програмного забезпечення, збільшує гнучкість та взаємозамінність компонентів. З боку бізнесу – це скорочення витрат на розробку та підтримку логічно об’ємних програм, зниження ризиків людської помилки, легка адаптивність та швидка розгортка. Зі сторони користувача – економія часу, використання єдиного (схожого) інтерфейсу для взаємодії, який буде вже зручним та інтуїтивно зрозумілим.

Якщо досліджувати український ринок, то саме українські філіали IT компаній розроблюють значну частину корпоративних застосунків та задовольняють велику кількість потреб сучасної комерційної сфери.

1.2 Змістовний опис та аналіз предметної області

Архітектурне рішення для керування ліцензіями на програмне забезпечення – це система, призначена для допомоги постачальникам програмного забезпечення та організаціям у керуванні доступами на їх

програмні продукти. Він надає функції для збереження, моніторингу, оптимізації та розповсюдження ліцензій, забезпечують дотримання ліцензійних угод.

Не зважаючи на те, що існує велика кількість готових програмних рішень, жодне з них не має відкритого коду, який можна вільно змінювати, масштабувати та перевикористовувати. Тому сучасній бізнес-індустрії бракує уніфікованого, легко змінюваного та інтегрованого модулю, щоб використовувати при розробці. Модуля, для налаштування та розгортки якого, не знадобиться велика команда розробників різних спеціалізацій.

1.3 Огляд існуючих рішень

На сьогоднішній день розробляється та використовується велика кількість сервісів та додатків для керування ліцензіями з різноманітними архітектурами та програмними підходами. Залежно від масштабу застосування, деякі сервіси є великими проектами, які оброблюють тисячі запитів щосекунди, інші спроектовані, як малі рішення для виконання локальних задач. У результаті, програмне забезпечення використовує зовсім різні технології.

Проаналізуємо відоме на сьогодні програмне забезпечення у даній області та технічні рішення, що допоможуть у реалізації модуля для обробки та зберігання ліцензій. Далі будуть розглянуті допоміжні засоби та готові програмні рішення.

1.3.1 Критерії для аналізу існуючих рішень

Під час розробки, метою інженерів програмного забезпечення є реалізація рішення з оптимальним відношенням ціни, що включає в себе як розробку, так і підтримку, до вихідного функціоналу.

Зазвичай оцінити якість сервісу можна за допомогою наступних критеріїв:

– ціна використання;

– легкість розгортки та налаштування;

– легкість інтеграції;

– стабільність продукту у стресових умовах;

– можливість заміни та розширення функціоналу компонентів.

1.3.2 FlexNet Manager Suite

FlexNet Manager Suite від Flexera — це комплексне рішення для керування активами програмного забезпечення. Допомагає організаціям керувати ліцензіями, відстежувати їх використання і оптимізувати витрати.

Переваги:

– комплексне рішення з багатьма розширеними функціями для керування активами програмного забезпечення та оптимізації ліцензій;

– повна звітність про використання, дозволяючи організаціям визначити ліцензії, що недостатньо використовуються, та оптимізувати витрати на програмне забезпечення;

– надає функціонал для відстеження виконання організацією умов ліцензійної угоди.

Недоліки:

– складне запровадження і налаштування, потребує виділення значних ресурсів та експертизи;

– висока вартість рішення, порівняно варіантами конкурентами, особливо для малих організацій з обмеженими бюджетами.

1.3.3 Sentinel RMS

Sentinel RMS (система керування доступами) від Thales — це гнучке рішення для керування ліцензіями, яке підтримує різноманітні моделі ліцензування. Це дозволяє постачальникам програмного забезпечення контролювати, керувати та контролювати використання ліцензій.

Переваги:

– підтримує широкий спектр моделей ліцензування, що забезпечує гнучкість для постачальників програмного забезпечення.;

– пропонує розширені механізми контролю ліцензій, щоб запобігти несанкціонованому використанню;

– забезпечує можливість відстеження використання та підготовки звітів для контролю використання ліцензій.

Недоліки:

– вимагає глибоких технічних навичок для впровадження та налаштування;

– важкий та обмежений, у порівнянні з іншими рішеннями, програмний інтерфейс та інтерфейс користувача.

1.3.4 Open LM

OpenLM — це рішення для керування ліцензіями, розроблене спеціально для організацій, що займаються розробкою програмного забезпечення. Він допомагає керувати та оптимізувати ліцензії на програмне забезпечення від різних постачальників, відстежувати використання ліцензій і забезпечити виконання ліцензійної згоди.

Переваги:

– спеціально розроблений сервіс для інженерів та організацій, які займаються розробкою програмного забезпечення, врахувуючи їх унікальні потреби у ліцензуванні.;

– надає функції для моніторингу, оптимізації використання ліцензій та для керування виконанням ліцензійної згоди;

– підтримує інтеграцію з різними постачальниками програмного забезпечення та менеджерами ліцензій.

Недоліки:

– вимагає глибоких технічних навичок для налаштування та підтримки;

– обмежена підтримка неінженерних галузей програмного забезпечення;

– необхідність великої кількості додаткових налаштувань або для використання розширених функцій.

1.3.5 Keygen

Keygen — це хмарна платформа керування ліцензіями, яка надає API та інструменти для створення, активації та керування ліцензіями. Він пропонує гнучкі моделі ліцензування та функції для постачальників програмного забезпечення для контролю та відстеження існуючих ліцензій.

Переваги:

– хмарна платформа з розширеним програмним інтерфейсом та інструментами легкої інтеграції;

– гнучкість налаштування моделей ліцензування;

– розробнико-орієнтовний підхід та велика документація.

Недоліки:

– хмарне рішення потребує стабільного підключення до мережі інтернет для активації та керування ліцензіями;

– деякі організації можуть віддавати перевагу локальним базам для забезпечення більшого рівня безпеки.

Таблиця 1.1 – Порівняльна таблиця

Критерій	FlexNet	Sentinel RMS	Open LM	Keygen	Власне рішення
Ціна	висока	середня	середня	низька	низька
Розгортка	швидка	довга	довга	швидка	швидка
Розширення	-	-	-	+	+
Стабільність	висока	висока	середня	середня	висока
Програмний інтерфейс	-	-	+	+	+

Таким чином, найбільш близьким за змістом та функціоналом є застосунок Keygen, який також використовує хмарні сервіси та архітектуру.

1.4 Аналіз вимог до програмного забезпечення

Після проведеного аналізу, на виконання даної роботи було поставлено завдання реалізувати інтегрований модуль, легко масштабований та легко розгортуваний модуль зі зручним програмним інтерфейсом.

Даний продукт повинен таким вимогам:

- хмарна мікросервісна архітектура;
- кросплатформність
- легка масштабованість;
- швидка та зручна розгортка для будь-якого проекту;
- низька зв’язність компонентів;
- розширений програмний інтерфейс;
- можливість інтегрування користувачів зі стороннім сервісами;
- гнучкість у редагуванні;
- можливість інтеграції через HTTP.

Діаграма варіантів використання представлена у графічному додатку.

В таблицях 1.2 – 1.25 наведені варіанти використання програмного забезпечення.

Таблиця 1.2 – Варіант використання UC-1

Use case name	Get License by ID
Use case ID	UC-01
Goals	Отримати сутність ліцензії за її ID
Actors	User
Trigger	Користувач бажає переглянути ліцензію
Pre-conditions	-
Flow of Events	Користувач робить запит до License API та вказує ID необхідної йому ліцензії.
Extension	У випадку введення ID у некоректному форматі, то воно підсвічується червоним і запит не створюється.
Post-Condition	API робить запит до відповідного сховища.

Таблиця 1.3 – Варіант використання UC-2

Use case name	Delete License
Use case ID	UC-02
Goals	Видалити сутність ліцензії за її ID
Actors	User
Trigger	Користувач бажає видалити ліцензію
Pre-conditions	-
Flow of Events	Користувач робить запит до License API та вказує ID необхідної йому ліцензії.
Extension	У випадку введення ID у некоректному форматі, то воно підсвічується червоним і запит не створюється.
Post-Condition	API перевіряє, чи існує ця ліцензія, та робить запит до відповідного сховища.

Таблиця 1.4 – Варіант використання UC-3

Use case name	Update license
Use case ID	UC-03
Goals	Оновити сутність ліцензії
Actors	User
Trigger	Користувач бажає оновити ліцензію

Продовження таблиці 1.4

Pre-conditions	-
Flow of Events	Користувач робить запит до License API та вказує сутність, яку необхідно змінити.
Extension	Якщо ліцензія оновлює продукт, на який посилається, то надсилається запит до Products API для первірки наявності такого продукту.
Post-Condition	API перевіряє, чи існує ця ліцензія, та робить запит до відповідного сховища.

Таблиця 1.5 – Варіант використання UC-4

Use case name	Create License
Use case ID	UC-04
Goals	Створити сутність ліцензії
Actors	User
Trigger	Користувач бажає створити ліцензію
Pre-conditions	-
Flow of Events	Користувач робить запит до License API, вказуючи ID продукту та параметри ліцензії. Надсилається запит до Products API, задля перевірки існування такого продукту.
Extension	-
Post-Condition	API робить запит до відповідного сховища.

Таблиця 1.6 – Варіант використання UC-5

Use case name	Check if product exists
Use case ID	UC-05
Goals	Перевірити існування продукту
Actors	User
Trigger	Користувач бажає перевірити існування продукту
Pre-conditions	Користувач намагається створити чи оновити ліцензію
Flow of Events	Запит надсилається на Products API
Extension	-
Post-Condition	Products API ідентифікує існування такого продукту

Таблиця 1.7 – Варіант використання UC-6

Use case name	Check if license exists
Use case ID	UC-06
Goals	Перевірити існування ліцензії
Actors	User
Trigger	Користувач бажає перевірити існування ліцензії
Pre-conditions	Користувач намагається видалити ліцензію
Flow of Events	До сховища надсилається запит на отримання ліцензії
Extension	-
Post-Condition	License API ідентифікує існування такої ліцензії

Таблиця 1.8 – Варіант використання UC-7

Use case name	Get user by UUID
Use case ID	UC-07
Goals	Отримати сутність користувача
Actors	User
Trigger	Користувач бажає знайти профіль в системі
Pre-conditions	-
Flow of Events	Надсилається запит до інтегрованого стороннього сервісу обробки користувачів
Extension	-
Post-Condition	Сторонній сервіс повертає сутність, якщо така існує

Таблиця 1.9 – Варіант використання UC-8

Use case name	Create user
Use case ID	UC-08
Goals	Створити нового користувача в системі
Actors	User
Trigger	Користувач бажає створити новий профіль в системі
Pre-conditions	-
Flow of Events	Генерується запит на створення нового користувача та надсилається до SNS топіку.
Extension	-
Post-Condition	SNS отримує новий запис.

Таблиця 1.10 – Варіант використання UC-9

Use case name	Update user
Use case ID	UC-09
Goals	Оновити користувача в системі
Actors	User
Trigger	Користувач бажає оновити профіль в системі
Pre-conditions	-
Flow of Events	Генерується запит на оновлення та надсилається до SNS.
Extension	-
Post-Condition	SNS отримує новий запис.

Таблиця 1.11 – Варіант використання UC-10

Use case name	Delete user
Use case ID	UC-10
Goals	Видалити користувача з системи
Actors	User
Trigger	Користувач бажає видалити профіль з системи
Pre-conditions	-
Flow of Events	Генерується запит на видалення користувача та надсилається до SNS топіку.
Extension	-
Post-Condition	SNS отримує новий запис.

Таблиця 1.12 – Варіант використання UC-11

Use case name	Push to SNS Topic
Use case ID	UC-11
Goals	Надіслати повідомлення до SNS топіку
Actors	User
Trigger	Система намагається надіслати запит про створення, оновлення чи видалення користувача
Pre-conditions	Користувач намагається створити, оновити чи видалити профіль
Flow of Events	Система розміщує запит у SNS сервісі
Extension	-
Post-Condition	SNS оброблює та зберігає новий запис

Таблиця 1.13 – Варіант використання UC-12

Use case name	Send to SQS subscribers
Use case ID	UC-12
Goals	Надіслати запит всім підписаним сервісам
Actors	User
Trigger	Система надсилає збережений запит усім підписаним на SNS чергам
Pre-conditions	Користувач намагається створити, оновити чи видалити профіль
Flow of Events	SNS оброблює та зберігає новий запис
Extension	-
Post-Condition	SNS надсилає запит усім підписаним на неї сервісам

Таблиця 1.14 – Варіант використання UC-13

Use case name	Get message from queue
Use case ID	UC-13
Goals	Отримати запит про зміну користувача
Actors	User
Trigger	Сервіс отримав SQS повідомлення від SNS топіку
Pre-conditions	Користувач намагається створити, оновити чи видалити профіль
Flow of Events	SNS оброблює та зберігає новий запис
Extension	Якщо стан Circuit Breaker'а: - "відкритий" – повідомлення повертається назад у чергу; - "закритий" – повідомлення відправляється на обробку - "напіввідкритий" – сервіс намагається опрацювати повідомлення. У разі невдачі стан переходить у "відкритий", у разі успіху – у "закритий"
Post-Condition	Запит у SQS черзі оброблено

Таблиця 1.15 – Варіант використання UC-14

Use case name	Process message
Use case ID	UC-14

Продовження таблиці 1.15

Goals	Обробити запит з черги
Actors	User
Trigger	Система по інтеграції користувачів зчитує інформацію про запит
Pre-conditions	Користувач намагається створити, оновити чи видалити профіль
Flow of Events	Система інтегрує користувача зі стороннім сервісом
Extension	Якщо кількість несталих помилок більше, ніж заданий ліміт, то стан Circuit Breaker'а переходить у "відкритий" на час таймауту, а повідомлення повертається до черги
Post-Condition	Юзер успішно інтегрований у сторонній сервіс

Таблиця 1.16 – Варіант використання UC-15

Use case name	Open
Use case ID	UC-15
Goals	Перевести Circuit Breaker у "відкритий" стан
Actors	User
Trigger	Система прораховує новий статус Circuit Breaker'а
Pre-conditions	Кількість несталих помилок більше за ліміт, невдала обробка запиту у "напіввідкритому" стані
Flow of Events	Сервіс вимикає підписку на SQS чергу на час таймауту, планується повідомлення про перехід стану після часу таймауту
Extension	Якщо підписку вже вимкнено, або повідомлення вже заплановано, то ці дії не виконуються
Post-Condition	Стан Circuit Breaker переходить у "відкритий"

Таблиця 1.17 – Варіант використання UC-16

Use case name	Half Open
Use case ID	UC-16
Goals	Перевести Circuit Breaker у "напіввідкритий" стан
Actors	User
Trigger	Система прораховує новий статус Circuit Breaker'а
Pre-conditions	Сервіс отримав системне повідомлення про зміну стану

Продовження таблиці 1.17

Flow of Events	Сервіс намагається опрацювати одне повідомлення з черги, у разі невдачі стан переходить у “відкритий”; у разі успіху стан переходить у “закритий”
Extension	Якщо підписку вже вимкнено, або повідомлення вже заплановано, то ці дії не виконуються
Post-Condition	Стан Circuit Breaker переходить у “напіввідкритий”

Таблиця 1.18 – Варіант використання UC-17

Use case name	Get Product
Use case ID	UC-17
Goals	Отримати сутність продукту
Actors	User
Trigger	Користувач бажає отримати продукт
Pre-conditions	-
Flow of Events	API надсилає відповідний запит до системи
Extension	-
Post-Condition	Система отримала запит про пошук продукту

Таблиця 1.19 – Варіант використання UC-18

Use case name	Delete Product
Use case ID	UC-18
Goals	Видалити сутність продукту
Actors	User
Trigger	Користувач бажає видалити продукт
Pre-conditions	-
Flow of Events	API надсилає відповідний запит до системи
Extension	-
Post-Condition	Система отримала запит про видалення продукту

Таблиця 1.20 – Варіант використання UC-19

Use case name	Create Product
Use case ID	UC-19
Goals	Створити сутність продукту

Продовження таблиці 1.20

Actors	User
Trigger	Користувач бажає створити продукт
Pre-conditions	-
Flow of Events	API надсилає відповідний запит до системи
Extension	-
Post-Condition	Система отримала запит про створення продукту

Таблиця 1.21 – Варіант використання UC-20

Use case name	Update Product
Use case ID	UC-20
Goals	Оновити сутність продукту
Actors	User
Trigger	Користувач бажає оновити продукт
Pre-conditions	-
Flow of Events	API надсилає відповідний запит до системи
Extension	-
Post-Condition	Система отримала запит про оновлення продукту

Таблиця 1.22 – Варіант використання UC-21

Use case name	Get Product Entitlement
Use case ID	UC-21
Goals	Створити права на продукт
Actors	User
Trigger	Користувач бажає створити права на продукт
Pre-conditions	-
Flow of Events	Система шукає ліцензію, права на яку намагається створити користувач, шукає пов’язаний з нею продукт та завантажує відповідні деталі. Потім, перевіряє чи існує такий користувач, завантажує деталі користувача та створює відповідну сутність. Сутність надсилається до системи
Extension	Якщо ліцензії, продукта чи користувача не існує, то повертається помилка
Post-Condition	Система отримала запит про створення права на продукт

Таблиця 1.23 – Варіант використання UC-22

Use case name	Update Product Entitlement
Use case ID	UC-22
Goals	Оновити права на продукт
Actors	User
Trigger	Користувач бажає оновити право на продукт
Pre-conditions	-
Flow of Events	Якщо оновлюється ліцензія, то система перевіряє її, шукає пов’язаний з нею продукт та завантажує відповідні деталі. Потім, перевіряє чи існує такий користувач, завантажує деталі користувача та створює відповідну сутність. Сутність надсилається до системи
Extension	Якщо ліцензії, продукта чи користувача не існує, то повертається помилка
Post-Condition	Система отримала запит про пошук продукту

Таблиця 1.24 – Варіант використання UC-23

Use case name	Get Product Entitlement
Use case ID	UC-23
Goals	Отримати права на продукт
Actors	User
Trigger	Користувач бажає отримати сутність права на продукт
Pre-conditions	-
Flow of Events	API надсилає відповідний запит до системи
Extension	-
Post-Condition	Система отримала запит про пошук права на продукт

Таблиця 1.25 – Варіант використання UC-24

Use case name	Delete Product Entitlement
Use case ID	UC-24
Goals	Видалити права на продукт
Actors	User
Trigger	Користувач бажає видалити сутність права на продукт
Pre-conditions	-
Flow of Events	API надсилає відповідний запит до системи

Продовження таблиці 1.25

Extension	-
Post-Condition	Система отримала запит про видалення права на продукт

1.4.1 Розроблення функціональних вимог

Функціональні вимоги сервісу представлено у таблицях: 1.25 – 1.34

Таблиця 1.25 – Функціональна вимога FR-1

Назва	Зміна ліцензії
Опис	Користувач може створювати, видаляти та змінювати ліцензії

Таблиця 1.26 – Функціональна вимога FR-2

Назва	Зміна продукту
Опис	Користувач може створювати, видаляти та змінювати продукти

Таблиця 1.27 – Функціональна вимога FR-3

Назва	Зміна профілю користувача
Опис	Користувач може створювати, видаляти та змінювати профілі користувача

Таблиця 1.28 – Функціональна вимога FR-4

Назва	Зміна права на продукту
Опис	Користувач може створювати, видаляти та змінювати права на продукт

Таблиця 1.29 – Функціональна вимога FR-5

Назва	Інтеграція користувачів
Опис	Система повинна інтегрувати користувачів зі стороннім сервісом

Таблиця 1.30 – Функціональна вимога FR-6

Назва	Відмовостійкість
Опис	Система повинна зберігати невдало оброблені запити по зміні користувачів

Таблиця 1.31 – Функціональна вимога FR-7

Назва	Отримання ліцензії
Опис	Користувач може отримувати ліцензії

Таблиця 1.32 – Функціональна вимога FR-8

Назва	Отримання продукту
Опис	Користувач може отримувати продукти

Таблиця 1.33 – Функціональна вимога FR-9

Назва	Отримання профіля користувача
Опис	Користувач може отримувати профілі користувачів

Таблиця 1.34 – Функціональна вимога FR-10

Назва	Отримання прав на продукт
Опис	Користувач може отримувати права на продукт

На рисунку 1.1 представлена матриця трасування вимог.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10
UC-01							+			
UC-02	+									
UC-03	+									
UC-04	+									
UC-05		+						+		
UC-06	+						+			
UC-07									+	
UC-08			+							
UC-09			+							
UC-10			+							
UC-11			+			+				
UC-12			+			+				
UC-13			+			+				
UC-14			+			+				
UC-15						+				
UC-16						+				
UC-17								+		
UC-18		+								
UC-19		+								
UC-20		+								
UC-21				+						
UC-22				+						
UC-23										+
UC-24				+						

Рисунок 1.1 – Матриця трасування вимог

1.4.2 Розроблення нефункціональних вимог

Програма повинна відповідати таким нефункціональним вимогам:

– додаток не має використовувати фізичних ресурсів, а тому має знаходитися у хмарних сервісах;

– інфраструктура має бути описана за допомогою IaC інструментів;

– наявність CI/CD пайплайну для швидкої розгортки та налаштування;

– час відповіді не повинен перевищувати 1 секунду;

– додаток повинен створювати, зберігати та маніпулювати ліцензіями продуктів;

– програмне забезпечення має бути легко масштабованим, компоненти легко замінені;

– елементи програми повинні мати лише той доступ, який необхідний для функціонування;

– усі помилки мають бути перехоплені та оброблені;

– система має відповідати усім чинним законам і нормам;

– наявність розширеної документації та механізму логування.

1.5 Постановка задачі

Робота над додатком – створення уніфікованого шаблону хмарного сервісу, який буде відповідати за взаємодію між користувачами та ліцензіями, вирішити проблему масштабованості такого застосунку та використання на всіх сучасних операційних системах.

Для того, щоб досягти даного результату, було поставлено наступні задачі:

– розробити механізм автоматичного відновлення системи у разі збою third-party системи;

– розробити механізм інтеграції застосунку зі сторонніми сервісами;

- система має зберігати, створювати, оновлювати та видаляти користувачів;
- система має зберігати, створювати, оновлювати та видаляти продукти;
- система має зберігати, створювати, оновлювати та видаляти ліцензії;
- реалізувати CI/CD механізм для швидкої та зручної розгортки застосунку.

Для сервісу було поставлено наступні вимоги:

- можливість швидкого та зручного зв’язку дані між користувачем та застосунком;
- легка масштабованість проекту;

- зручна розгортка без потреби у великій команді розробників;
- відповідність за керування ліцензіями продуктів чи послуг користувача.

Кінцева програма повинна виконувати всі дані пункти та реалізувати увесь відповідний функціонал. Вона повинна реагувати на дії користувача та відповідати заданим вимогам.

Висновки до розділу

У цьому розділі була визначена предметна область теми, проведений її змістовний опис та аналіз, визначені функціональні і нефункціональні вимоги.

Було визначено повний стек технологій нашого проекту: хмарний застосунок з використанням Amazon Web Services, мікро сервісна архітектура на платформі .NET (C#) та використанням Terraform і Jenkins у якості CI/CD інструментів.

Аналіз існуючих рішень показав, що аналогам не вистачає стабільності, адекватної цінової політики та інструментів розробника для інтегрування і розширення модулів. Головні функції застосунку були винесені у діаграму використання.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Основною задачею під час розробки програмного модуля стало вирішення проблеми інтеграції компонентів сервісу з іншим програмним забезпеченням та сторонніми (third-party) сервісами. У зв’язку з тим, що робота цих сервісів не залежить від працездатності модуля, необхідно розробити механізм, який буде контролювати доступність низхідних (downstream) сервісів. У якості вирішення був обраний архітектурний патерн Circuit Breaker (далі – перемикач) та реалізовано для використання AWS компонентів. Circuit Breaker – це патерн проектування, який застосовується в розподілених системах для керування збоїв і помилок у взаємодії між компонентами. Головна мета – забезпечити стабільність і надійність системи шляхом зниження впливу збоїв в одній частині системи на інші. Перемикач буде блокувати обробку повідомлень на час, заданий параметром тайм-аут.

Основні причини застосування патерну перемикач:

– захист від збоїв: у разі помилки перемикач швидко переключається у "відкритий" стан, уникаючи запитів до некоректної частини системи, не поширюючи помилки на всю систему;

– зменшення часу відновлення: швидка зміна стану перемикача дозволяє уникнути затримок, пов'язаних з запитами, які неможливо опрацювати, і дозволяє системі швидше адаптуватись до збою;

– контроль навантаження: перемикач дозволяє контролювати навантаження на систему. Він може встановлювати обмеження на кількість запитів, що допомагає запобігти перевантаженню;

– деградація замість відмови: замість повної відмови у відповіді на запити під час збою, перемикач може надати альтернативні шляхи обробки з більшою інформативністю та обмеженою функціональністю. Це дозволяє

системі продовжувати працювати, навіть якщо з частиною її компонентів виникли проблеми;

– моніторинг та керування: перемикач надає інструменти для моніторингу стану системи та виявлення помилок і збоїв, шляхом реєстрації помилок та автоматичним переходом між станами.

На рисунку 2.1 представлена діаграма станів перемикача.

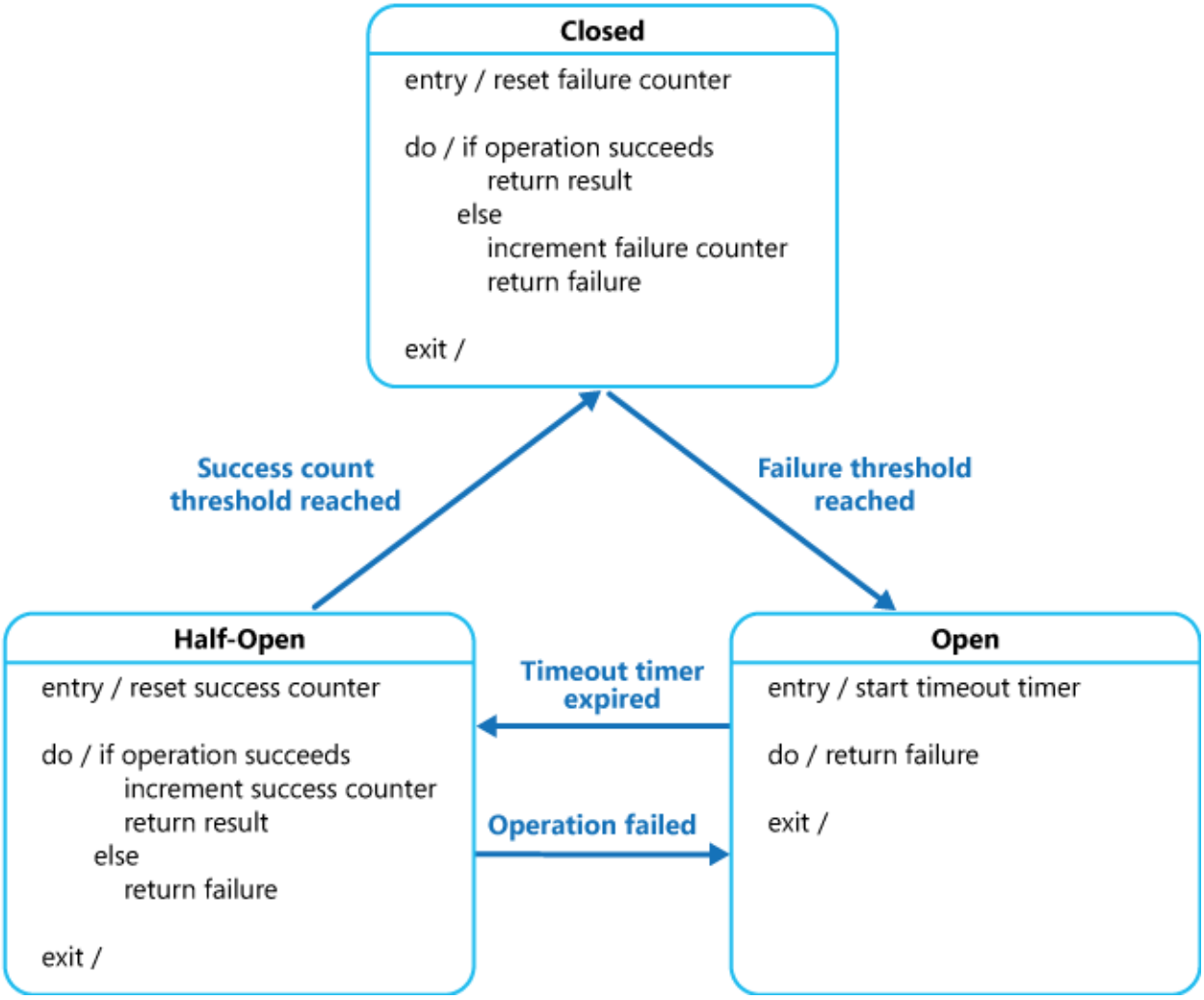


Рисунок 2.1 – Діаграма станів перемикача

Лямбда функція отримує повідомлення з черги, на яку підписана. Якщо сервіс, від якого лямбда залежить, недоступний, то перемикач заблокує обробку повідомлень задля економії ресурсів, часу та коштів на обчислювальні потужності.

Діаграма послідовності роботи перемикача наведена у графічних додатках.

2.2 Архітектура програмного забезпечення

Розроблюваний застосунок буде використовувати мікро сервісну архітектуру, де кожен сервіс представлено у вигляді AWS лямбда функції, який відповідає за обробку принципово різних сутностей.

License Management Lambda виконує функцію по роботі з ліцензіями та призначення цих ліцензій відповідним користувачам (сутність власної ліцензії має назву право на продукт). Ця лямбда працює з двома відповідними БД таблицями – Licenses та ProductsEntitlements. Всі логи виконання надходять до відповідного сховища у AWS.

Product Management Lambda маніпулює продуктами в системі: створює, видаляє, оновлює та повертає деталі відповідних сутностей. Ця лямбда працює з БД таблицею Products. Всі логи виконання надходять до відповідного сховища у AWS.

Users Management Lambda виконує функцію створення користувачів в системі та формуванням повідомлень про зміну відповідним стороннім сервісам. Також, ця лямбда функція відповідає за повернення деталей необхідного системі користувача. Ця лямбда працює з БД таблицею Users лише на зчитування. Всі логи виконання надходять до відповідного сховища у AWS.

Users Integration Lambda створена для того, щоб інтегрувати користувачів зі сторонніми сервісами, який в цій роботі представлений базою даних Users. Саме користувачі частіше за всього мають бути інтегровані зі сторонніми сервісами, тому для функціонування цієї лямбди був реалізований патерн перемикач. У випадку неможливості з’єднання зі стороннім сервісом, перемикач припиняє обробку повідомлень з черги задля економії обчислювальних потужностей і мінімізації можливості втрати запитів. Втрата запиту зі створення, оновлення чи видалення користувача може призвести до фатальних наслідків і великих грошових втрат, тому використання перемикача у даному випадку надважливе. Всі запити при цьому залишаються у черзі і

чекають на час їх обробки. Через деякий час налаштований планувальний надсилає повідомлення, щоб спробувати опрацювати повідомлення з черги. Якщо операція успішна, то перемикач закривається і обробка продовжується в штатному режимі. Всі логи виконання надходять до відповідного сховища у AWS. На рисунку 2.2 представлена діаграма компонентів перемикача.

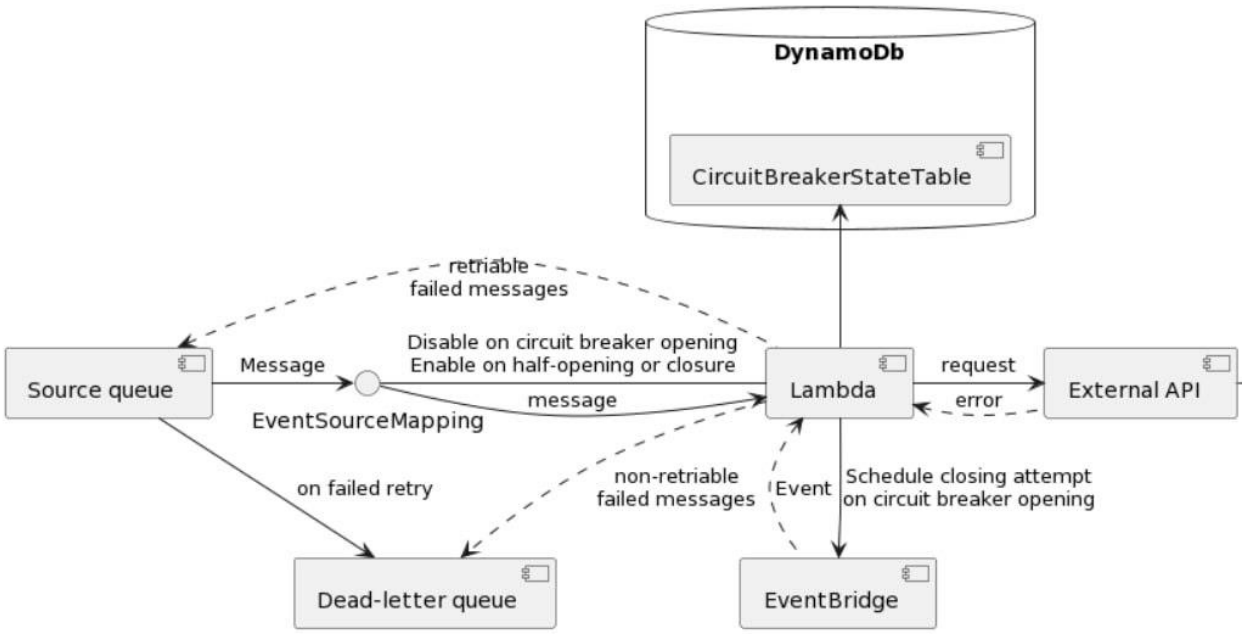


Рисунок 2.2 – Діаграма компонентів перемикача

Загально, кожен сервіс розділено на три умовні шари: domain, infrastructure та views. Рівень Views є нічим іншим як рівнем, що відповідає за взаємодію з вебом. Він приймає HTTP запити, робить валідацію та викликає бізнес логіку. Наступним після views шаром є domain - це бізнес логіка. При моделюванні застосунку, було використано головне правило проектування – політики (бізнес правила) не повинні залежати від деталей. Детамями є веб фреймворк (ASP.NET), є бази даних (Dynamo DB). Клієнт для взаємодії може змінюватись, тому логіка повинна бути чітко розділена. Рівень даних, складається з Repositories – це класи, що інкапсулюють в собі деталі зберігання даних, таким чином, що бізнес логіка не залежить від бази даних.

Кожен мікро сервіс є відокремленим один від одного і може спілкуватися лише завдяки HTTP запитам. Необхідно створити систему,

компоненти якої можуть швидко змінюватись, замінятись або вилучатись взагалі, тому єдиним має залишатися лише програмний інтерфейс для взаємодії.

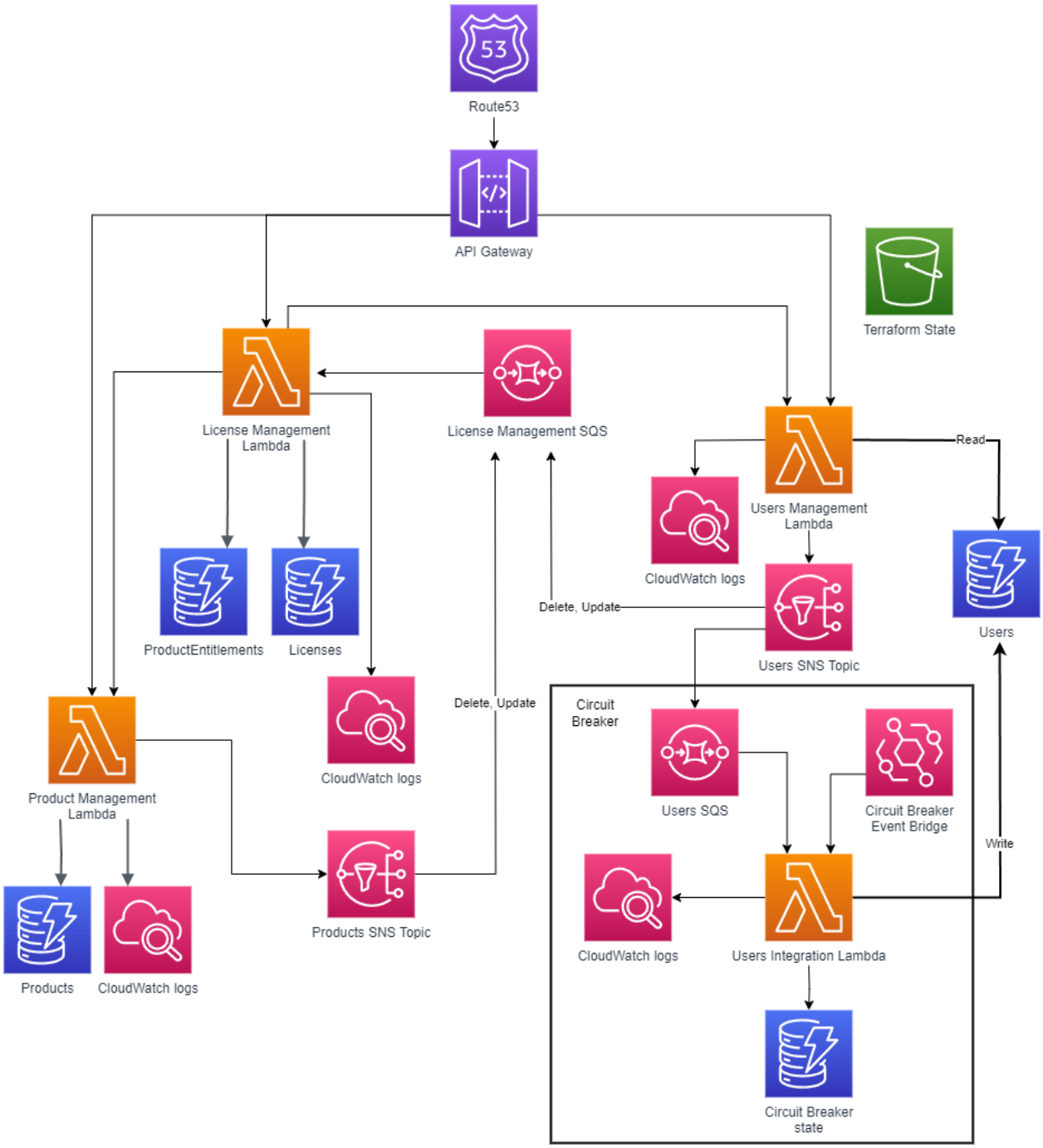


Рисунок 2.3 – Архітектура програмного модуля

2.3 Конструювання програмного забезпечення

Детальний опис класів та функцій наведений в таблицях 2.1 – 2.2

Таблиця 2.1 – Опис класів додатку

Клас	Опис
BaseMessage	Базовий клас для сутності SNS повідомлення
LicenseCreateModel	Модель для створення ліцензії
LicenseDto	Сутність для збереження ліцензії у БД
ProcessAction	Перелік можливих маніпуляцій з сутністю
ProductDto	Сутність для збереження продукту у БД
ProductEntitlementDto	Сутність для збереження права на продукт у БД
UserDto	Сутність для збереження користувача у БД
EntitlementNotFoundException	Помилка, що свідчить про неіснуюче право на продукт
LicenseNotFoundException	Помилка, що свідчить про неіснуючу ліцензію
ProductNotFoundException	Помилка, що свідчить про неіснуючий продукт
UserNotFoundException	Помилка, що свідчить про неіснуючого користувача
MappingStateTransitionException	Помилка, що свідчить про невдалу зміну стану Event Source Mapping
DatabaseConfigurationExtensions	Містить розширення для конфігурації DynamoDB таблиць

Продовження таблиці 2.1

ParametersConfigurationExtensions	Містить розширення для ініціалізації змінних середовища лямбди
SnsConfigurationExtensions	Містить розширення для налаштування доступу до Simple Notification Service
SwaggerConfigrationExtensions	Містить розширення для налаштування Swagger XML документації
LicenseMapper	Клас для конвертації моделі ліцензії у сутніть бази даних
UnhandledExceptionLoggingMiddleware	Компонент ASP.NET Core middleware для налаштування логування помилок
EventBridgeOptions	Містить в собі необхідні параметри для налаштування AWS Event Bridge
EventBridgeSchedulerService	Сервіс для роботи з AWS Event Bridge Schedule
SnsClient	Клієнт для роботи з AWS Simple Notification Service
SqsClient	Клієнт для роботи з AWS Simple Queue Service
LambdaContextAccessor	Клас-обгортка для доступу до контексту лямбди
CircuitBreakerPolicy	Представляє політику виконання перемикача

Продовження таблиці 2.1

CircuitState	Абстрактний клас для представлення стану перемикача
ClosedState	Описує закритий стан перемикача
ClosedStateConfig	Містить усі необхідні налаштування для закритого стану перемикача
ExecutionResult	Описує результат виконання виклику перемикачем
HalfOpenState	Описує напівзакритий стан перемикача
OpenCircuitException	Помилка, що свідчить про відкритий стан перемикача на даний момент часу
OpenState	Описує відкритий стан перемикача
OpenStateConfig	Містить усі необхідні налаштування для відкритого стану перемикача
PermanentlyClosedState	Описує постійно закритий стан перемикача
LicenseController	Контролер для API ліцензій
ProductEntitlementController	Контролер для API прав на продукти
LambdaParameters	Містить налаштування змінних середовища лямбди
LicenseRepository	Репозиторій для роботи з БД ліцензій
ProductEntitlementRepository	Репозиторій для роботи з БД прав на продукти
LicenseManagementService	Сервіс для обробки ліцензій
ProductEntitlementManagementService	Сервіс для обробки прав на продукти

Продовження таблиці 2.1

ProductController	Контролер для API продуктів
ProductManagementService	Сервіс для обробки продуктів
ProductRepository	Репозиторій для роботи з БД продуктів
ServiceProviderBuilder	Клас для налаштування базових залежностей для функціонування застосунку
NoSuitableHandlerException	Помилка, що свідчить про відсутність обробника для вхідних даних
CircuitBreakerConfigurationExtensions	Містить розширення для налаштування перемикача
CircuitBreakerMessageHandlerStrategy	Стратегія з обробки системного повідомлення перемикача
DataHandlerStrategySelector	Відбірник стратегії для вхідних даних
UserIntegrationHandlerStrategy	Стратегія з обробки повідомлення по інтеграції користувача
CircuitBreakerActions	Перелік можливих дій з перемикачем
CircuitBreakerMessage	Сутність системного повідомлення перемикача
CircuitStateDatabaseDto	Сутність для збереження стану перемикача у БД
CircuitBreakerOptions	Містить налаштування перемикача для змінних середовища лямбди
CircuitStateRepository	Репозиторій для роботи з БД стану користувача

Продовження таблиці 2.1

UsersWriteRepository	Репозиторій для запису та зміни даний у БД користувачів
CircuitBreakingService	Сервіс, що інтегрує перемикач із застосунком
CircuitStateToDatabaseDtoMapper	Клас для конвертації стану користувача з сутності БД у модель
SqsEventSourceMappingClient	Клієнт для роботи з lambda event source mapping
SqsEventProcessingService	Сервіс з обробки SQS подій
SqsRecordProcessingService	Сервіс з обробки SQS повідомлень
UserIntegrationHandler	Сервіс для інтеграції користувача зі сторонніми сервісами
UserValidator	Містить правила валідації для сутності користувача
Function	Точка входу в лямбда функцію
UsersController	Контролер для API користувачів
UsersReadRepository	Репозиторій для зчитування користувачів з БД
UserManagementService	Сервіс з обробки користувачів
LambdaEntryPoint	Точка входа в веб апі лябмда функцію
LocalEntryPoint	Використовується для локального запуску лямбда функції у вигляді веб апі

Таблиця 2.2 – Опис методів основних класів додатку

Клас	Метод	Опис

Продовження таблиці 2.2

DatabaseConfiguration Extensions	ConfigureDynamoDB	Налаштовує доступ лямбда функції до БД
LoggerConfiguration Extensions	ConfigureLogging	Налаштовує логування роботи лямбда функції
ParametersConfiguration Extensions	ConfigureLambda Variables	Налаштовує змінні середовища лямбда функції
SnsConfiguration Extensions	ConfigureSns	Налаштовує взаємодію з Simple Notification Service
SwaggerConfigration Extensions	ConfigureSwagger Services	Налаштовує свагер документацію
SwaggerConfigration Extensions	UseSwagger	Налаштовує свагер інтерфейс
LicenseMapper	MapToDto	Конвертує модель ліцензії у сутність бази даних
EventBridgeScheduler Service	Cancel CircuitClosureTrial	Відміняє планування системного повідомлення
EventBridgeScheduler Service	Schedule CircuitClosureTrial	Планує час системного повідомлення
SnsClient	PublishToTopicAsync	Відправляє повідомлення до SNS топіку
SqsClient	Enqueue	Надсилає повідомлення до SQS черги

Продовження таблиці 2.2

SqsClient	Dequeue	Завантажує повідомлення з SQS черги
SqsClient	DeleteMessage	Видаляє повідомлення з SQS черги
CircuitBreakerPolicy	Handle	Налаштовує список помилок для реагування перемикачем
CircuitBreakerPolicy	Execute	Робить виклик до стороннього сервісу
CircuitBreakerPolicy	ClosePermanently	Переводить стан перемикача у постійно закритий
CircuitState	Accept	Повертає сутність стану перемикача
CircuitState	RecordError	Оброблює помилку залежно від стану
CircuitState	ExecuteAsync	Оброблює виклик до стороннього сервісу
LicenseController	GetLicense	Робить запит на повернення ліцензії
LicenseController	CreateLicense	Робить запит на створення ліцензії
LicenseController	UpdateLicense	Робить запит на оновлення ліцензії
LicenseController	DeleteLicense	Робить запит на видалення ліцензії

Продовження таблиці 2.2

ProductEntitlement Controller	GetEntitlement	Робить запит на повернення права на продукт
ProductEntitlement Controller	CreateEntitlement	Робить запит на створення права на продукт
ProductEntitlement Controller	DeleteEntitlement	Робить запит на видалення права на продукт
ProductEntitlement Controller	UpdateEntitlement	Робить запит на оновлення права на продукт
IReadRepository	GetByIdAsync	Повертає сутність з відповідної БД
IWriteRepository	DeleteAsync	Видаляє сутність з відповідної БД
IWriteRepository	SaveAsync	Зберігає сутність у відповідну БД
LicenseManagement Service	CreateLicense	Оброблює логіку створення ліцензії
LicenseManagement Service	DeleteLicense	Оброблює логіку видалення ліцензії
LicenseManagement Service	GetLicenseById	Оброблює логіку повернення ліцензії
LicenseManagement Service	UpdateLicense	Оброблює логіку оновлення ліцензії

Продовження таблиці 2.2

ProductEntitlement ManagementService	CreateEntitlement	Оброблює логіку створення права на продукт
ProductEntitlement ManagementService	DeleteEntitlement	Оброблює логіку видалення права на продукт
ProductEntitlement ManagementService	GetEntitlementById	Оброблює логіку повернення права на продукт
ProductEntitlement ManagementService	UpdateEntitlement	Оброблює логіку оновлення права на продукт
ProductController	GetProduct	Робить запит на отримання продукту
ProductController	CreateProduct	Робить запит на створення продукту
ProductController	UpdateProduct	Робить запит на оновлення продукту
ProductController	DeleteProduct	Робить запит на видалення продукту
ProductManagement Service	CreateProduct	Оброблює логіку створення продукту
ProductManagement Service	DeleteProduct	Оброблює логіку видалення продукту
ProductManagement Service	GetProductById	Оброблює логіку повернення продукту
ProductManagement Service	UpdateProduct	Оброблює логіку оновлення продукту

Продовження таблиці 2.2

CircuitBreakerConfiguration Extensions	Configure CircuitBreakerServices	Налаштовує заложності, необхідні для повноцінної роботи перемикача
IDataHandlerStrategy	IsSuitable	Перевіряє чи дана стратегію може обробити вхідні дані
IDataHandlerStrategy	ProcessInput	Оброблює вхідні дані
DataHandlerStrategy Selector	GetStrategy	Повертає стратегію, яка може обробити вхідні дані
CircuitStateRepository	GetAsync	Повертає поточний стан перемикача
CircuitStateRepository	SaveAsync	Зберігає поточний стан перемикача
CircuitBreakingService	ExecuteAsync	Оброблює повідомлення з черги та змінює стан перемикача
CircuitBreakingService	Close	Змінює стан перемикача на закритий
CircuitBreakingService	HalfOpen	Змінює стан перемикача на напіввідкритий
CircuitBreakingService	Open	Змінює стан перемикача на відкритий

Продовження таблиці 2.2

CircuitBreakingService	PermanentlyClose	Змінює стан перемикача на постійно закритий
CircuitBreakingService	Trial	Завантажує повідомлення з черги та намагається обробити його
CircuitState ToDatabaseDtoMapper	Visit	Конвертує сутність стану перемикача з БД у модель
SqsEventSourceMapping Client	Disable EventSourceMapping	Вимикає виклик лямбда функції від SQS черги
SqsEventSourceMapping Client	Enable EventSourceMapping	Вмикає трігер лямбда функції від SQS черги
SqsEventProcessing Service	ProcessAsync	Оброблює SQS Event з черги
SqsRecordProcessing Service	ProcessMessage	Оброблює SQS повідомлення
SqsRecordProcessing Service	ProcessSqsRecords	Оброблює всі записи у івенті
UserIntegrationHandler	CreateUser	Створює користувача у сторонньому сервісі
UserIntegrationHandler	DeleteUser	Видаляє користувача зі стороннього сервіса
UserIntegrationHandler	UpdateUser	Оновлює користувача у сторонньому сервісі
UserManagementService	CreateUser	Оброблює логіку генерації запиту на створення користувача

Продовження таблиці 2.2

UserManagementService	DeleteUser	Оброблює логіку генерації запиту на видалення користувача
UserManagementService	GetUserByUserName	Оброблює логіку генерації запиту на повернення користувача за іменем
UserManagementService	GetUserByUuid	Оброблює логіку генерації запиту на повернення користувача за ID
UserManagementService	UpdateUser	Оброблює логіку генерації запиту на оновлення користувача
UsersReadRepository	GetByIdAsync	Повертає користувача з БД із заданим ID
UsersReadRepository	GetByUsernameAsync	Повертає користувача з БД із заданим ім’ям

Бібліотеки, сервіси та ПЗ, що було використано під час виконання роботи представлено у таблиці 2.3

Таблиця 2.3 – Використані інструменти

Назва використаного ПЗ	Опис застосування
Visual Studio 2022	Середовище розробки програмного забезпечення від компанії Microsoft .
Visual Studio Code	Текстовий редактор.

Продовження таблиці 2.3

Terraform	Інструмент для створення, зміни та відстеження версій інфраструктури за допомогою спеціального синтаксису та коду.
Jenkins	Програмна система з відкритим вихідним кодом, побудований на Java Virtual Machine (JVM), що підтримує тисячі плагінів для розробки, розгортання та автоматизації програмного забезпечення.
Serilog	Механізм логування з широкими можливостями налаштування.
Swashbuckle	NuGet пакет, що вбудовує у WebAPI автогенерацію інформації про вузли відповідно до специфікації OpenAPI.
AWS Simple Notification Service	Сервісний пакет AWS, який забезпечує обмін повідомленнями у від'єднаних мікросервісах, безсерверних обчислювальних системах та інших розподілених системах.
AWS Lambda	AWS Lambda – це сервіс serverless обчислень, який запускає програмний код у відповідь на певні події та відповідає за автоматичне виділення необхідних обчислювальних ресурсів.
AWS Simple Queue Service	Повністю керована черга повідомлень для мікросервісів, розподілених систем та безсерверних додатків.
AWS Event Bridge	Сервіс, що надає в режимі реального часу доступ до змін даних в AWS, власних програмах та програмах SaaS без написання коду.

Продовження таблиці 2.3

AWS API Gateway	Повністю керований сервіс для розробників, який спрощує публікацію, обслуговування, моніторинг, захист та використання API у будь-яких масштабах.
AWS Route 53	Високо доступний та масштабований веб-сервіс системи доменних імен (DNS).
AWS Dynamo DB	Повністю керована безсерверна база даних NoSQL на основі пар «ключ-значення», створена для запуску високопродуктивних програм у будь-якому масштабі. Має вбудований захист, безперервне резервне копіювання, автоматичну реплікацію в кількох регіонах, кешування в пам'яті та інструменти імпорту та експорту даних.
AWS Identity and Access Management	Сервіс для налаштування безпечного керування ідентифікаційними даними та доступом до ресурсів AWS
AWS S3	Сервіс для зберігання об'єктів, що пропонує найкращі в галузі показники продуктивності, масштабованості, доступності та безпеки даних.
AWS Console	Веб-додаток для адміністрування ГІС-серверів та інших ресурсів, створених на AWS.

DynamoDB – це NoSQL база даних, тому модель БД представлена чотирма таблицями, які не пов’язані одна з одною. На таблицях 2.4 – 2.7 наведено опис цих таблиць.

Таблиця 2.4 – Опис таблиці Users

Назва поля	Тип даних	Унікальність	Опис
Uuid	Guid	Так	Унікальний ідентифікатор користувача
Username	String	Так	Ім’я профілю
Title	String	Ні	Звертання (містер/місіс)
Firstname	String	Ні	Ім’я
Lastname	String	Ні	Фамілія
PhoneNumber	String	Ні	Номер телефону
EmailAddress	String	Ні	Електронна адреса
PrefferedServiceLanguage	String	Ні	Мова використання застосунку
Type	String	Ні	Тип (особа/сервіс)
UpdatedOn	DateTime Offset	Ні	Дата останнього оновлення

Таблиця 2.5 – Опис таблиці Products

Назва поля	Тип даних	Унікальність	Опис
ProductId	Guid	Так	Унікальний ідентифікатор продукту

Продовження таблиці 2.5

Name	String	Hi	Назва продукту
Description	String	Hi	Опис продукту
ProductGroup	String	Hi	Група продуктів
Billable	String	Hi	Платний
Countries	String	Hi	Країни розповсюдження

Таблиця 2.6 – Опис таблиці Licenses

Назва поля	Тип даних	Унікальність	Опис
LicenseId	Guid	Так	Унікальний ідентифікатор ліцензії
ProductId	Guid	Так	Ідентифікатор ліцензованого продукту
PriceAmount	Decimal	Hi	Ціна ліцензії
Currency	String	Hi	Грошова валюта

Таблиця 2.7 – Опис таблиці Entitlements

Назва поля	Тип даних	Унікальність	Опис
EntitlementId	Guid	Так	Унікальний ідентифікатор права на продукт
LicenseId	Guid	Так	Ідентифікатор ліцензії

Продовження таблиці 2.7

UserId	Guid	Hi	Ідентифікатор користувача
ProductName	String	Hi	Назва продукту
ProductDescription	String		Опис продукту
UserName	String		Ім’я користувача
Until	DateTime		Термін активації

На рисунку 2.4 представлено схему бази даних програмного забезпечення.

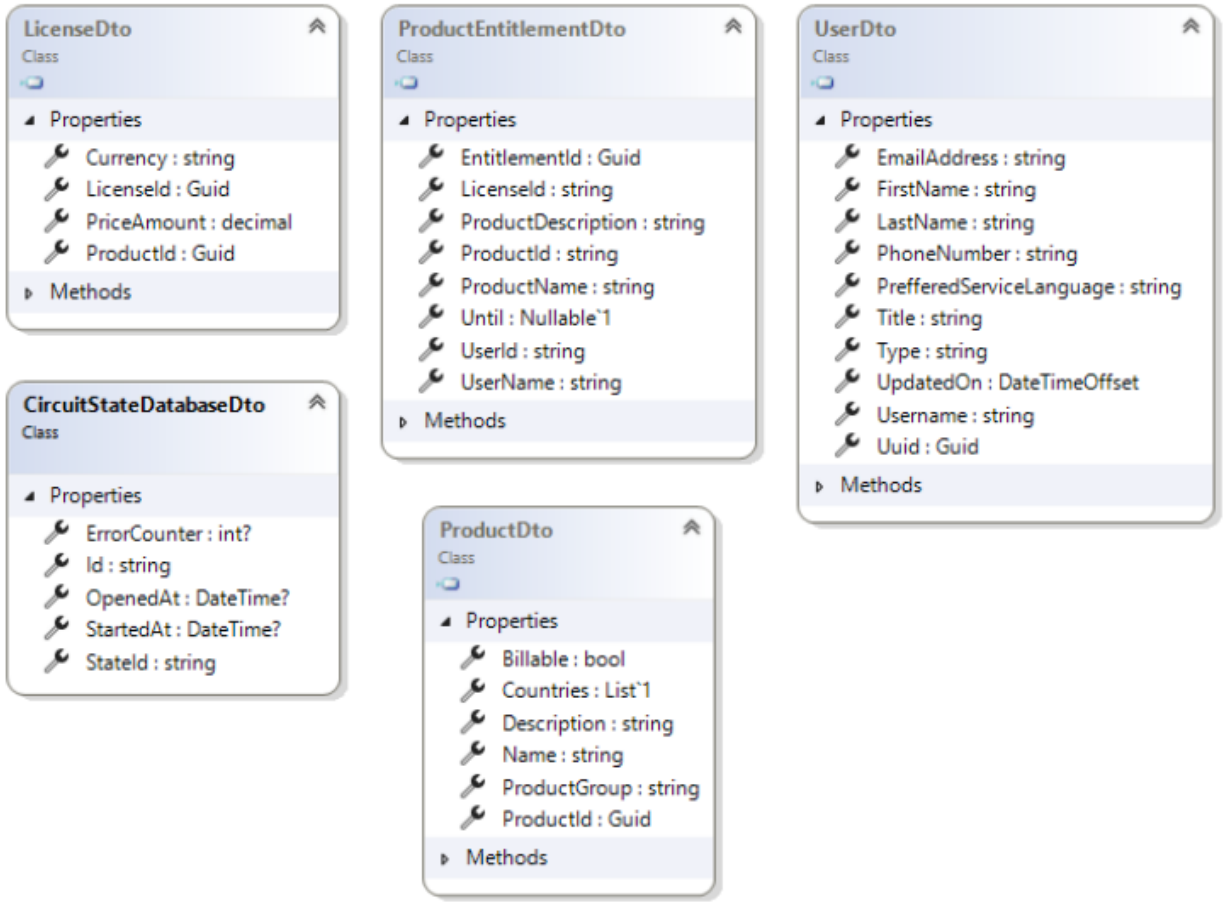


Рисунок 2.4 – Схема бази даних системи

2.4 Аналіз безпеки даних

Захист даних у програмній реалізації забезпечують AWS сервіси.

Загальний доступ до ресурсів застосунку налаштовується за допомогою Identity and Access Management (AWS IAM). IAM контролює, щоб у компонентів програми були лише ті доступи, які необхідні для повноцінного функціонування. IAM надає можливості двофакторної аутентифікації та інших механізми безпеки для захисту облікових записів користувачів.

Цілісність і збереження даних контролюється AWS Dynamo DB за допомогою механізмів шифрування, які захищають від несанкціонованого доступу. Динамічна масштабованість дозволяє автоматично збільшувати потужність для обробки збільшеної навантаження, а можливість створювати резервні копії даних забезпечують відновлення у разі випадкового видалення або втрати.

Дані, надіслані за допомогою AWS Simple Notification Service шифруються з використанням AWS Key Management Service (KMS). Ідентифікація та автентифікація виконується за допомогою ключів доступу та політик безпеки IAM. Є можливість використання точки доступу VPC для обмеження доступу до SNS на внутрішню мережу.

AWS Simple Queue service шифрує повідомлення з використанням KMS для захисту конфіденційності даних. Доступ до черг керується за допомогою IAM політик.

API Gateway налаштований за допомогою політики аутентифікації та авторизації. Можливе також використання SSL/TLS для шифрування передачі даних через API. Наявний захист від атак типу DDoS.

Route 53 забезпечує систему можливістю настроювання брандмауерів та шифрування даних з використанням SSL/TLS.

Вихідний код лямбда функцій захищено інструментами Amazon Web Services та доступний тільки користувачам гіт репозиторію.

Варто відзначити, що належна конфігурація та налаштування цих сервісів є важливими для забезпечення безпеки даних. Рекомендується ретельно налаштовувати політики доступу IAM, використовувати шифрування для даних у спокої, захищати API та забезпечувати надійний контроль доступу до всіх використовуваних сервісів.

Висновки до розділу

У другому розділі були розглянуті та проаналізовані наступні пункти:

– деталі архітектурної реалізації;

– системні вимоги до ПЗ;

– взаємодія різних рівнів програмного забезпечення;

– класи та методи компонентів;

– сутності бази даних;

– утиліти та бібліотеки необхідні для розробки ПЗ;

– аналіз безпеки користування застосунком.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Для аналізу якості програмного забезпечення можна використовувати такі ключові метрики:

1) час виявлення та виправлення дефектів: більш швидке виявлення та виправлення дефектів може свідчити про ефективність процесу розробки та здатність системи швидко реагувати на виняткові ситуації або адаптуватися до них.

За останній місяць було виявлено 9 дефектів під час тестування, середній час виправлення помилок – 3 години;

2) розмір програмного коду: обсяг програмного коду, зазвичай в рядках (LOC). Кількість рядок використовуються для оцінки складності проекту та його обсягу.

LOC – 4846;

3) сумісність: можливість та ефективність роботи на різних платформах, операційних системах, браузерах тощо. Може включати показники сумісності з різними версіями програмного або апаратного забезпечення.

Даний застосунок сумісний з будь-якою операційною системою та браузером, оскільки є хмарним сервісом;

4) відсоток повторного використання коду: Ця метрика вимірює, яка частина коду може бути повторно використана у різних частинах програми або в інших проектах. Використання повторного коду сприяє зменшенню зусиль розробки, покращенню якості та забезпеченню більшої стабільності.

8.3 % коду є повторно використаним у різних модулях проекту;

5) складність коду: цикломатична складність (Cyclomatic Complexity), кількість вузлів прийняття рішень (Decision Points), визначає складність логіки програми та появу логічних помилок.

Середня цикломатична складність – 4;

6) надійність та доступність: час безвідмовної роботи (MTBF – Mean Time Between Failures), час відновлення після відмови (MTTR – Mean Time to Recover), час відновлення після аварійного відмови (MTTF – Mean Time to Failure) та інші.

MTBF – 153 години.

MTTR – 1.2 секунди.

MTTF – 1 година.

3.2 Опис процесів тестування

План тестування наведено у таблицях 3.1 – 3.9.

Таблиця 3.1 – TC001

Назва	Опрацювати продукт
Передумови	Користувач бажає отримати, створити, оновити чи видалити продукт. Задано сутність, яка проходить правила валідації.
Дія	Сервіс робить запит до сховища і робить відповідні зміни.
Очікуваний результат	Продукт отримано, створено, оновлено чи видалено.

Таблиця 3.2 – TC002

Назва	Синхронізувати користувача
Передумови	Користувач бажає створити, оновити чи видалити профіль користувача. Задано сутність, яка проходить правила валідації.

Продовження таблиці 3.2

Дія	Сформувати запит на зміну чи створення сутності користувача та надіслати до інтеграційної лямбди. Повідомлення надходить в чергу і оброблюється.
Очікуваний результат	Профіль користувача створено, оновлено чи видалено, профіль синхронізовано з усіма сторонніми сервісами.

Таблиця 3.3 – ТС003

Назва	Завантажити профіль користувача
Передумови	Користувач бажає завантажити профіль користувача. Ідентифікатор вказано на сутність, що існує в системі.
Дія	Сервіс робить запит до сховища і завантажує відповідний профіль.
Очікуваний результат	Профіль користувача завантажено з усіма його деталями.

Таблиця 3.4 – ТС004

Назва	Створити ліцензію
Передумови	Користувач бажає створити ліцензію. Вказано ідентифікатор продукту, що існує в системі. Задано сутність, яка проходить правила валідації.
Дія	Сервіс перевіряє, чи існує продукт в системі. Робиться запит в систему для зберігання нової сутності ліцензії.
Очікуваний результат	Нова ліцензія з’являється у сховищі в системі.

Таблиця 3.5 – ТС005

Назва	Оновити ліцензію

Продовження таблиці 3.5

Передумови	Користувач бажає оновити ліцензію. Вказано продукт, що існує в системі. Задано сутність, яка проходить правила валідації.
Дія	Сервіс перевіряє, чи існує продукт в системі. Робиться запит в систему для зберігання нової сутності ліцензії.
Очікуваний результат	Ліцензія оновлена та збережена в сховищі в системі.

Таблиця 3.6 – ТС006

Назва	Видалити ліцензію
Передумови	Користувач бажає видалити ліцензію. Ідентифікатор вказує на ліцензію, що існує в системі.
Дія	Робиться запит в систему для видалення сутності ліцензії.
Очікуваний результат	Ліцензія видаляється зі сховища в системі.

Таблиця 3.7 – ТС007

Назва	Створити право на продукт
Передумови	Користувач бажає створити право на продукт. Вказано ідентифікатор ліцензії та користувача, що існують в системі. Задано сутність, яка проходить правила валідації.
Дія	Сервіс перевіряє, чи існує ліцензія та користувач в системі. Завантажуються деталі користувача та продукту, пов’язаного з ліцензією. Робиться запит у систему для зберігання нової сутності права на продукт.

Продовження таблиці 3.7

Очікуваний результат	Нова сутність права на продукт з’являється у сховищі в системі.

Таблиця 3.8 – TC008

Назва	Оновити право на продукт
Передумови	Користувач бажає оновити право на продукт. Вказано ідентифікатор ліцензії та користувача, що існують в системі. Задано сутність, яка проходить правила валідації.
Дія	Сервіс перевіряє, чи існує ліцензія та користувач в системі. Завантажуються деталі користувача та продукту, пов’язаного з ліцензією. Робиться запит у систему для оновлення сутності права на продукт.
Очікуваний результат	Право на продукт оновлюється у сховищі в системі.

Таблиця 3.9 – TC009

Назва	Видалити чи завантажити право на продукт
Передумови	Користувач бажає видалити чи оновити право на продукт. Вказано ідентифікатор права на продукт, що існують в системі. Задано сутність, яка проходить правила валідації.
Дія	Робиться запит у систему для видалення/завантаження сутності права на продукт.
Очікуваний результат	Право на продукт завантажується/видаляється зі сховища в системі.

Результати тестування представлені у таблиці 3.10.

Таблиця 3.10 – Результати тестування

Код тесту	Назва тесту	Результат
TC001	Опрацювати продукт	Успіх
TC002	Синхронізувати користувача	Успіх
TC003	Завантажити профіль користувача	Успіх
TC004	Створити ліцензію	Успіх
TC005	Оновити ліцензію	Успіх
TC006	Видалити ліцензію	Успіх
TC007	Створити право на продукт	Успіх
TC008	Оновити право на продукт	Успіх
TC009	Видалити чи завантажити право на продукт	Успіх

3.3 Опис контрольного прикладу

Створимо нового користувача в системі. Можна не вказувати Uuid самотужки, система автоматично присвоїть згенерований ідентифікатор:

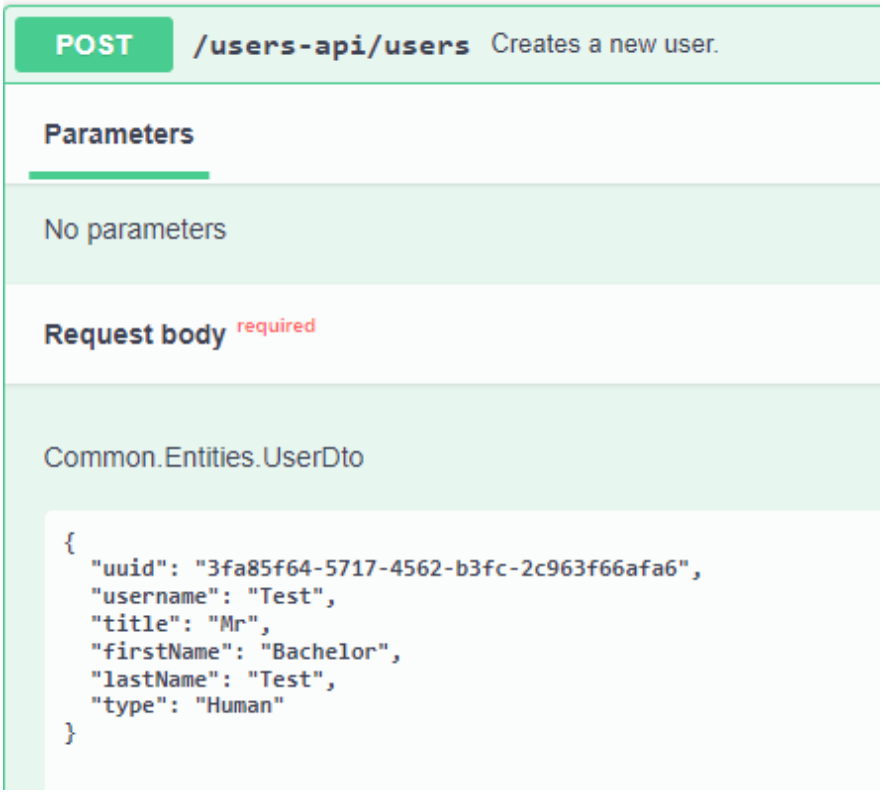


Рисунок 3.1 – Запит на створення нового користувача

Curl

```
curl -X 'POST' \
  'https://r2mu6cava6.execute-api.eu-central-1.amazonaws.com/dev/users-api/users' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "uuid": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "username": "Test",
  "title": "Mr",
  "firstName": "Bachelor",
  "lastName": "Test",
  "type": "Human"
}'
```

Request URL

```
https://r2mu6cava6.execute-api.eu-central-1.amazonaws.com/dev/users-api/users
```

Server response

Code	Details
202 *Undocumented*	Response body

```
{
  "uuid": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "username": "Test",
  "title": "Mr",
  "firstName": "Bachelor",
  "lastName": "Test",
  "phoneNumber": null,
  "emailAddress": null,
  "prefferedServiceLanguage": "en-US",
  "type": "Human",
  "updatedBy": null,
  "updatedOn": "2023-06-01T19:13:39.0668462+00:00"
}
```

Рисунок 3.2 – Результат виклику

Код 202 позначає, що запит на інтеграцію було прийнято. Перевіримо SQS чергу:

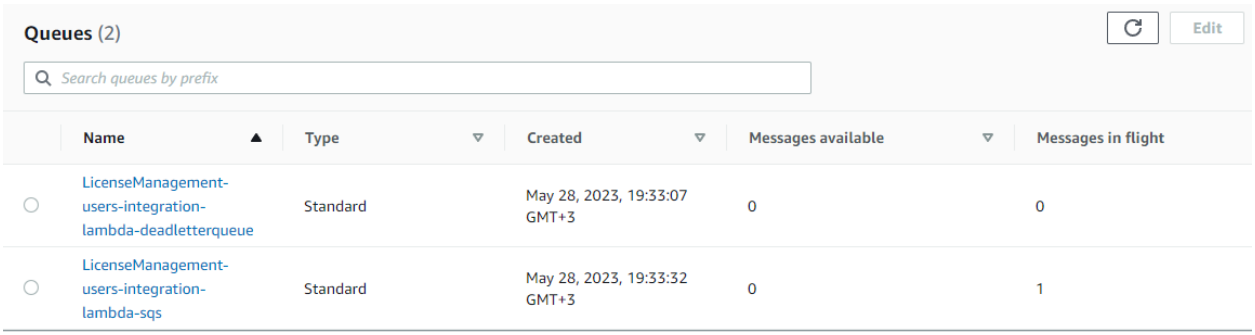


Рисунок 3.3 – SQS черга з інтеграції користувачів

На рисунку 3.3 видно, що нове повідомлення було сформовано та потрапило до черги з інтеграції користувачів. Перевіримо логи роботи інтеграційної лямбди:

Рисунок 3.4 – Отримане повідомлення

У повідомленні представлений ідентифікатор сутності, дія для обробки і деталі. Після отримання, лямбда функція створить користувача з тіла повідомлення та інтегрує його з усіма сторонніми сервісами (у нашому випадку це буде база даних). Перевіримо наявність нового користувача GET запитом:

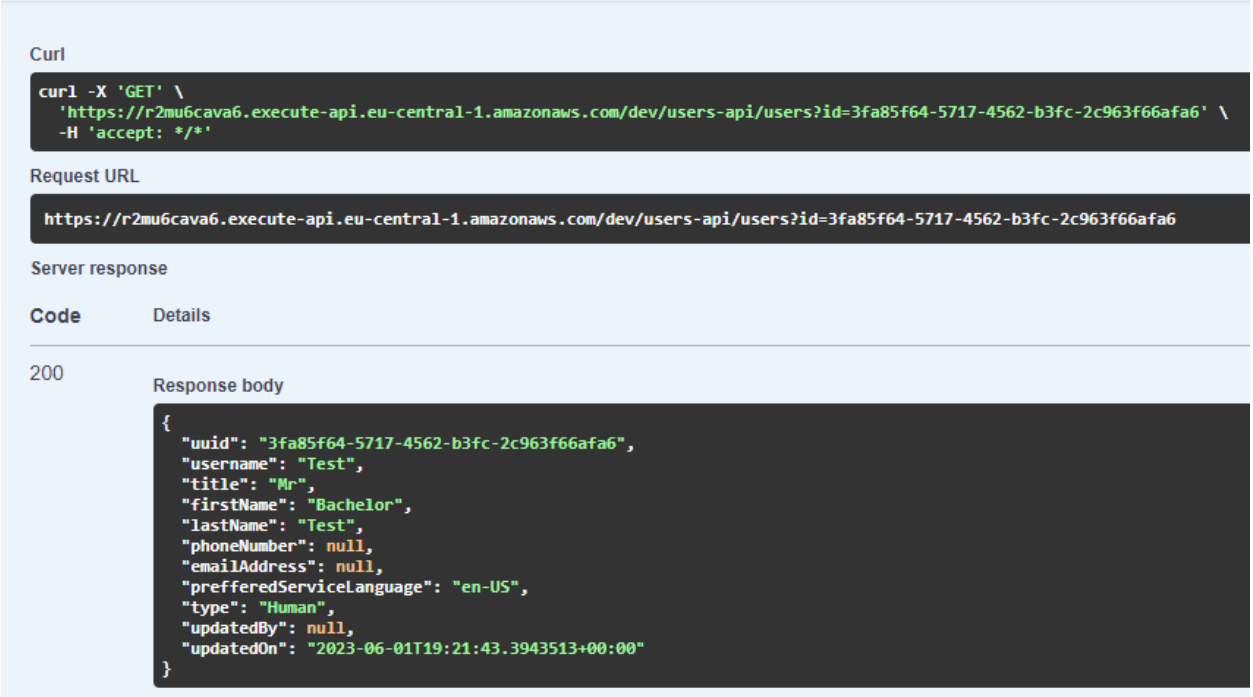


Рисунок 3.5 – Запит користувача з шуканим ідентифікатором

Користувач успішно створений. Порівняючи поля сутності, можна зробити висновки, що вона та ж сама.

Висновок: Очікуваний результат повністю співпадає з фактичним, тест пройдений успішно.

Висновки до розділу

У даному розділі було детально описано процес тестування застосунку. Представлені тест кейси, аналіз програмного забезпечення, згідно з метрик якості коду, різні можливі стани системи та контрольний приклад, який демонструє роботу інтеграції користувачів.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Окрему увагу під час розробки застосунку було виділено саме на зручну розгортку застосунку на будь-якій машині. Для цього уся необхідна хмарна інфраструктура була описана за допомогою IaC інструментів (Terraform у нашому випадку). Для створення ж цієї інфраструктури було створено CI/CD пайплайн за допомогою сервісу Jenkins.

Необхідно лише створити аккаунт у Jenkins та запустити інстанс на власній машині. Потім створити аккаунт в AWS та надати облікові дані у Jenkins Credentials Manager. Налаштувати шлях до репозиторію і запустити.

Результат запуску показано на рисунку 4.1.

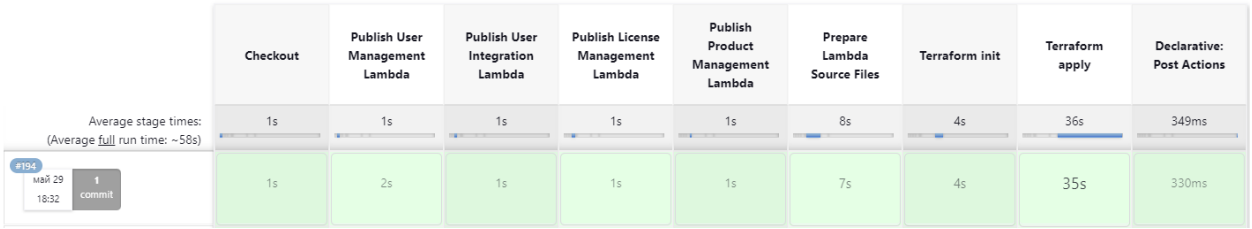


Рисунок 4.1 – Запуск Jenkins пайплайну

Спочатку буде завантажено вихідний код з репозиторію, створено тимчасову директорію, у яку будуть скомпільовані усі сервіси проекту. Збірки архівуються, шлях до них передається, як змінна у виклик команди terraform apply, яка перевіряє та створює усю інфраструктуру. У кінці тимчасова директорія видаляється незалежно від успіху попередніх етапів виконання.

Процес автоматичного розгортання програмного забезпечення за допомогою CI/CD пайплайнів займає близько хвилини для оновлення інфраструктуру, і близько 4 хвилин під час розгортки з нуля.

4.2 Підтримка програмного забезпечення

AWS надає велику кількість інструментів для підтримки програмного забезпечення, які були налаштовані протягом цієї роботи. Для забезпечення моніторингу та журналювання застосунку налаштовано інтеграцію з Amazon CloudWatch, який виконує логування виконання кожної лямбди. AWS X-Ray надає змогу для збору та відстеження метрик продуктивності та трасування запитів. Забезпечення масштабованості вашого застосунку може включати налаштування автоматичного масштабування для Lambda функцій, так щоб вони могли обробляти зростаюче навантаження. AWS Lambda Auto Scaling та AWS Application Auto Scaling, дозволяє автоматично налаштовувати кількість ресурсів для кожної функції залежно від потреб навантаження. За роутинг і його зміну відповідає API Gateway, за DNS – Route 53.

У разі необхідності внесення змін у AWS ресурси та інфраструктуру, необхідно знайти відповідний модуль у Terraform скриптах та вилучити або змінити його. За бажанням інфраструктура може бути розширена шляхом додавання нових модулів та ресурсів. Terraform автоматично виконує версіювання програмного забезпечення.

Вихідний код може бути вільно змінений та завантажений у новий репозиторій, якщо вказати новий шлях до нього у пайплайні. Під час розгортки, буде завантажена нова версія та замінена у лямбдах.

XML документація представлена описом усіх класів та методів додатка та генерується автоматично. Для сторонніх розробників створений окремий Swagger ендпоінт, що описує кожну Rest API, моделі запиту та відповіді та можливі повернені HTTP коди.

Висновки до розділу

У цьому розділі були показані інструкції, необхідні для впровадження та супроводу програмного забезпечення.

Були використані такі інструменти, як Jenkins та Terraform для зручної розгортки нашого додатку. Також, була описана послідовність дій у випадку оновлення, зміни чи розширення програмного застосунку.

ВИСНОВКИ

У результаті виконання роботи, був створений інтегрований модуль для обробки і збереження ліцензій. Проаналізовані існуючі аналоги та реалізації подібного програмного забезпечення на ринку. Описана соціально-економічна значущість роботи.

Були підкреслені можливі рішення з їх перевагами та недоліками, у результаті чого був зроблений висновок, щодо мінімального набору необхідних інструментів. Дана програма була розроблена за допомогою мови програмування C# на платформі .NET від компанії Microsoft, хмарного сервісу AWS (Amazon Web Services) та Terraform – інструмент від компанії Hashicorp, що допомагає декларативно керувати інфраструктурою. Проведено розбір архітектури, її складових частин та етапи розробки.

Після реалізації застосунок був протестований на пристроях з різними версіями операційних систем та браузерів, щоб переконатися у повноцінному функціонуванні додатку.

Програмне забезпечення працює швидко та забезпечує задовільну продуктивність, легко масштабується та розгортається, зручно інтегрується, адаптується до зміни навантаження. Процес розгортання та керування змінами був ефективним і забезпечив стабільність та надійність середовища. Були використані належні механізми захисту, налаштовані права доступу та шифрування конфіденційної інформації.

ПЕРЕЛІК ПОСИЛАНЬ

1) AWS resources documentation. [Електронний ресурс]. – 2023. Режим доступу до ресурсу: https://docs.aws.amazon.com/

2) CDK for Terraform. [Електронний ресурс]. – 2023. Режим доступу до ресурсу: https://developer.hashicorp.com/terraform

3) Terraform AWS provider. [Електронний ресурс]. – 2023. Режим доступу до ресурсу: https://registry.terraform.io/providers/hashicorp/aws/latest/docs

4) Circuit Breaker Pattern. [Електронний ресурс]. – 2023. Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker

5) Jenkins Official Documentation. [Електронний ресурс]. – 2023. Режим доступу до ресурсу: https://www.jenkins.io/doc/

6) Microservice architecture style. [Електронний ресурс]. – 2023. Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

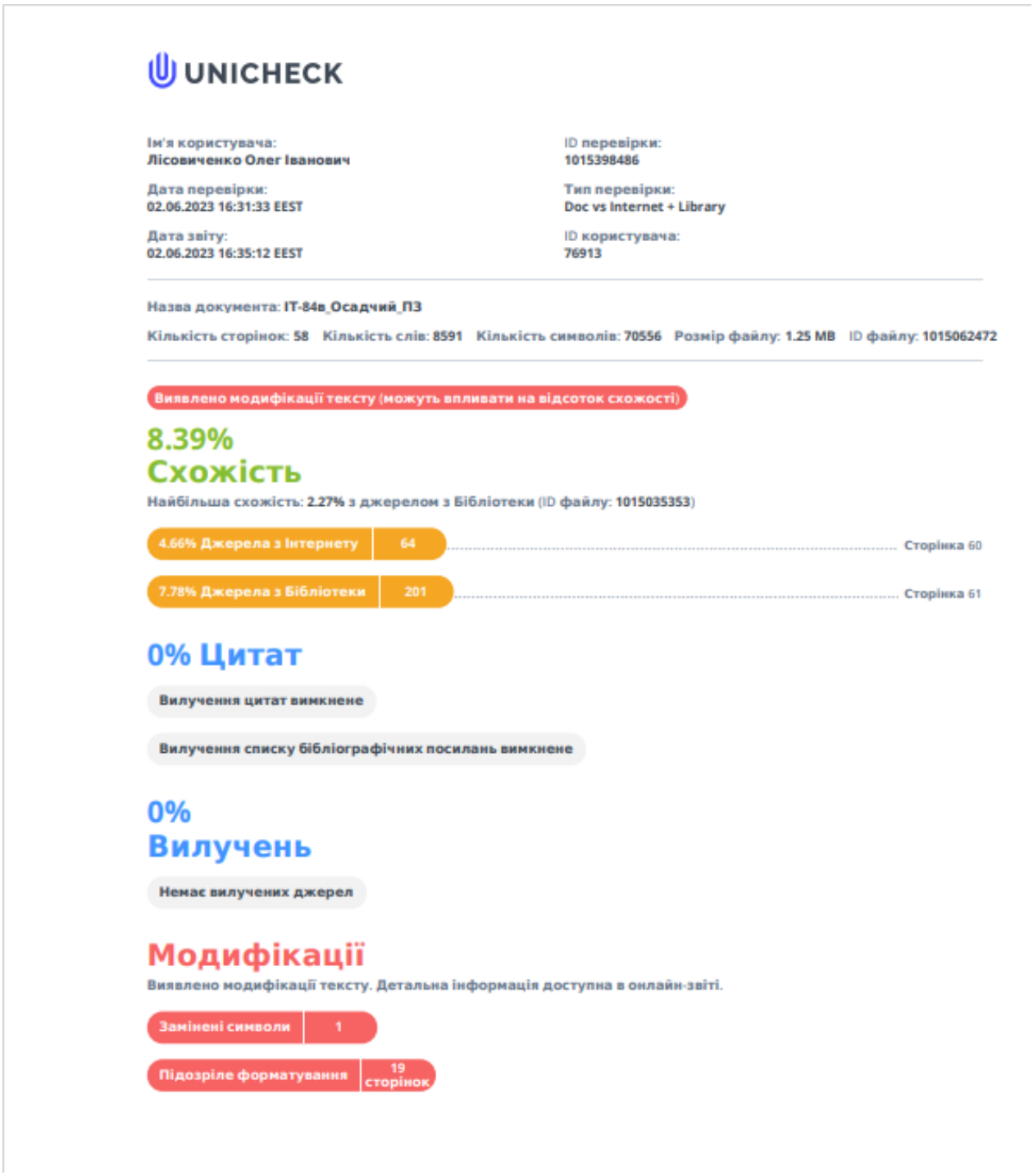
7) What is License Management Software. [Електронний ресурс]. – 2023. Режим доступу до ресурсу: https://cpl.thalesgroup.com/software-monetization/software-licensing-management

8) The Clean Architecture. [Електронний ресурс]. – 2012. Режим доступу до ресурсу: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

9) AWS SDK for .NET Version 3 API Reference. [Електронний ресурс]. – 2023. Режим доступу до ресурсу: https://docs.aws.amazon.com/sdkfornet/v3/apidocs/

10) Cloud Design Patterns. [Електронний ресурс]. – 2023. Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/architecture/patterns/

ДОДАТОК А. ЗВІТ ПОДІБНОСТІ.



UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015398486

Дата перевірки:
02.06.2023 16:31:33 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
02.06.2023 16:35:12 EEST

ID користувача:
76913

Назва документа: ІТ-84в_Осадчий_ПЗ

Кількість сторінок: 58 Кількість слів: 8591 Кількість символів: 70556 Розмір файлу: 1.25 MB ID файлу: 1015062472

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

8.39%
Схожість

Найбільша схожість: 2.27% з джерелом з Бібліотеки (ID файлу: 1015035353)

4.66% Джерела з Інтернету 64 ..... Сторінка 60

7.78% Джерела з Бібліотеки 201 ..... Сторінка 61

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 1

Підозріле форматування 19 сторінок

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ХМАРНЕ АРХІТЕКТУРНЕ РІШЕННЯ СЕРВІСУ ОБРОБКИ ЛІЦЕНЗІЙ ПРОДУКТІВ ТА ПОСЛУГ

Текст програми

КПІ. ІТ-84в23.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Мирослав ОСАДЧИЙ

Київ – 2023

Файл AwsEventSourceMappingState.cs

```
namespace UserIntegrationLambda.Services.CircuitBreaker
{
    public static class AwsEventSourceMappingState
    {
        public const string Enabled = "Enabled";
        public const string Disabled = "Disabled";

        public const string Enabling = "Enabling";
        public const string Disabling = "Disabling";
    }
}
```

Файл EntityTypes.cs

```
namespace Common.Constants
{
    public class EntityTypes
    {
        public const string User = "User";
        public const string Product = "Product";
        public const string License = "License";

        public const string Deleted = "[deleted]";
    }
}
```

Файл ValidationConstants.cs

```
namespace Common.Constants
{
    public static class ValidationConstants
    {
        public const string GuidRegexPattern = @"(?im)^[{(]?[0-9A-F]{8}[-]?(?:[0-9A-
F]{4}[-]?){3}[0-9A-F]{12}[)}]?$";
    }
}
```

Файл BaseMessage.cs

```
using Common.Interfaces;
using Newtonsoft.Json;
using Newtonsoft.Json.Converters;

namespace Common.Entities
{
    public class BaseMessage<T> : IMessage<T>
    {
        public BaseMessage(string entityId, string entityType, ProcessAction action)
        {
            EntityId = entityId;
            EntityType = entityType;
            Action = action;
        }

        public string EntityId { get; }

        public string EntityType { get; }

        [JsonConverter(typeof(StringEnumConverter))]
        public ProcessAction Action { get; }

        public DateTimeOffset MessageSentOn { get; set; } = DateTimeOffset.UtcNow;

        public T Content { get; set; } = default!;
```

```
        object IMessage.Content => Content!;
    }
}
```

Файл LicenseCreateModel.cs

```
namespace Common.Entities
{
    /// <summary>
    /// License model for create request.
    /// </summary>
    public class LicenseCreateModel
    {
        /// <summary>
        /// License cost value.
        /// </summary>
        public decimal PriceAmount { get; set; }

        /// <summary>
        /// Currency used to pay for license.
        /// </summary>
        public string? Currency { get; set; }
    }
}
```

Файл LicenseDto.cs

```
using Amazon.DynamoDBv2.DataModel;

namespace Common.Entities
{
    /// <summary>
    /// Represents license definition.
    /// </summary>
    [DynamoDBTable("LicenseManagement-Licenses")]
    public class LicenseDto
    {
        private string? currency;

        /// <summary>
        /// License's unique indentifier.
        /// </summary>
        [DynamoDBHashKey]
        public Guid LicenseId { get; set; } = Guid.NewGuid();

        /// <summary>
        /// Id of a product this license is assigned to.
        /// </summary>
        public Guid ProductId { get; set; }

        /// <summary>
        /// License cost value.
        /// </summary>
        public decimal PriceAmount { get; set; }

        /// <summary>
        /// Currency used to pay for license.
        /// </summary>
        public string Currency
        {
            get => currency ?? PriceAmount.ToString("C");
            set => currency = value;
        }
    }
}
```

Файл ProcessAction.cs

```
namespace Common.Entities
{
    /// <summary>
    /// Action being performed on an entity.
    /// </summary>
    public enum ProcessAction
    {
        /// <summary>
        /// Entity is being created.
        /// </summary>
        Create,

        /// <summary>
        /// Entity is being read.
        /// </summary>
        Read,

        /// <summary>
        /// Entity is being updated.
        /// </summary>
        Update,

        /// <summary>
        /// Entity is being deleted.
        /// </summary>
        Delete,

        /// <summary>
        /// Entity is being reinstated.
        /// </summary>
        Reinstate,

        /// <summary>
        /// Entity is being relocated.
        /// </summary>
        Relocate
    }
}
```

Файл ProductDto.cs

```
using Amazon.DynamoDBv2.DataModel;

namespace Common.Entities
{
    [DynamoDBTable("LicenseManagement-Products")]
    public class ProductDto
    {
        [DynamoDBHashKey]
        public Guid ProductId { get; set; } = Guid.NewGuid();

        public string Name { get; set; } = null!;

        public string Description { get; set; } = null!;

        public string ProductGroup { get; set; } = null!;

        public bool Billable { get; set; }

        public List<string> Countries { get; set; } = new() { "UA" };
    }
}
```

Файл ProductEntitlementDto.cs

```
using Amazon.DynamoDBv2.DataModel;

namespace Common.Entities
{
    [DynamoDBTable("LicenseManagement-Entitlements")]
    public class ProductEntitlementDto
    {
        [DynamoDBHashKey]
        public Guid EntitlementId { get; set; } = Guid.NewGuid();

        public string LicenseId { get; set; }

        public string UserId { get; set; }

        public string ProductName { get; set; } = null!;

        public string ProductDescription { get; set; } = null!;

        public string UserName { get; set; } = null!;

        public DateTime? Until { get; set; }
    }
}
```

Файл SqsEventSourceArn.cs

```
using System.ComponentModel.DataAnnotations;
using System.Text.RegularExpressions;

namespace Common.Entities
{
    public class SqsEventSourceArn
    {
        private const string SqsEventSourceArnPattern = @"^(arn):(aws):(sqs):[A-Za-
z0-9\-:]+";

        public string Value { get; }

        public SqsEventSourceArn([RegularExpression(SqsEventSourceArnPattern)]
string value)
        {
            if (value == null) throw new ArgumentNullException(nameof(value));

            if (!Regex.IsMatch(value, SqsEventSourceArnPattern)) throw new
ArgumentException($"{value} doesn't match {SqsEventSourceArnPattern} pattern.");

            Value = value;
        }

        public static bool IsMatch(string value)
        {
            return Regex.IsMatch(value, SqsEventSourceArnPattern);
        }
    }
}
```

Файл UserDto.cs

```
using Amazon.DynamoDBv2.DataModel;

namespace Common.Entities
{
    /// <summary>
    /// Represents model for user definition.
```

```
    /// </summary>
    [DynamoDBTable("LicenseManagement-Users")]
    public class UserDto
    {
        /// <summary>
        /// Unique-id created by the api processor.
        /// </summary>
        [DynamoDBHashKey]
        public Guid Uuid { get; set; } = Guid.NewGuid();

        /// <summary>
        /// Unique Id that a user logs in to their system with, often an email
address.
        /// </summary>
        public string Username { get; set; } = null!;

        /// <summary>
        /// Title of the user.
        /// </summary>
        public string? Title { get; set; }

        /// <summary>
        /// First name of a user.
        /// </summary>
        public string? FirstName { get; set; }

        /// <summary>
        /// Last name of a user.
        /// </summary>
        public string? LastName { get; set; }

        /// <summary>
        /// Phone number os a user/machine owner.
        /// </summary>
        public string? PhoneNumber { get; set; }

        /// <summary>
        /// Email address of a user/machine owner.
        /// </summary>
        public string? EmailAddress { get; set; }

        /// <summary>
        /// Preferred language services should use.
        /// </summary>
        public string PrefferedServiceLanguage { get; set; } = "en-US";

        /// <summary>
        /// User type (e.g Human, Service).
        /// </summary>
        public string Type { get; set; } = "Human";

        /// <summary>
        /// Last update date.
        /// </summary>
        public DateTimeOffset UpdatedOn { get; } = DateTimeOffset.UtcNow;
    }
}
```

Файл EntitlementNotFoundException.cs

```
using System.Runtime.Serialization;

namespace Common.Exceptions
{
    /// <summary>
```

```
    /// Represents an error during user retrieving operation.
    /// </summary>
    [Serializable]
    public class EntitlementNotFoundException : Exception
    {
        /// <summary>
        /// Initializes a new instance of <see cref="EntitlementNotFoundException"/>
class.
        /// </summary>
        public EntitlementNotFoundException() : this("Product Entitlement is not
found.")
        {
        }

        /// <summary>
        /// Initializes a new instance of <see cref="EntitlementNotFoundException"/>
class.
        /// </summary>
        /// <param name="message">Exception message.</param>
        public EntitlementNotFoundException(string? message) : base(message)
        {
        }

        /// <summary>
        /// Initializes a new instance of <see cref="EntitlementNotFoundException"/>
class.
        /// </summary>
        /// <param name="message">Exception message.</param>
        /// <param name="innerException">Inner exception.</param>
        public EntitlementNotFoundException(string? message, Exception?
innerException) : base(message, innerException)
        {
        }

        /// <summary>
        /// Initializes a new instance of <see cref="EntitlementNotFoundException"/>
class.
        /// </summary>
        /// <param name="info"><see cref="SerializationInfo"/></param>
        /// <param name="context"><see cref="StreamingContext"/></param>
        protected EntitlementNotFoundException(SerializationInfo info,
StreamingContext context) : base(info, context)
        {
        }
    }
}
```

Файл LicenseNotFoundException.cs

```
using System.Runtime.Serialization;

namespace Common.Exceptions
{
    /// <summary>
    /// Represents an error during license retrieving operation.
    /// </summary>
    [Serializable]
    public class LicenseNotFoundException : Exception
    {
        /// <summary>
        /// Initializes a new instance of <see cref="LicenseNotFoundException"/>
class.
        /// </summary>
        public LicenseNotFoundException() : this("License is not found.")
        {
```

```
        }

        /// <summary>
        /// Initializes a new instance of <see cref="LicenseNotFoundException"/>
class.
        /// </summary>
        /// <param name="message">Exception message.</param>
        public LicenseNotFoundException(string? message) : base(message)
        {
        }

        /// <summary>
        /// Initializes a new instance of <see cref="LicenseNotFoundException"/>
class.
        /// </summary>
        /// <param name="message">Exception message.</param>
        /// <param name="innerException">Inner exception.</param>
        public LicenseNotFoundException(string? message, Exception? innerException)
: base(message, innerException)
        {
        }

        /// <summary>
        /// Initializes a new instance of <see cref="LicenseNotFoundException"/>
class.
        /// </summary>
        /// <param name="info"><see cref="SerializationInfo"/></param>
        /// <param name="context"><see cref="StreamingContext"/></param>
        protected LicenseNotFoundException(SerializationInfo info, StreamingContext
context) : base(info, context)
        {
        }
    }
}
```

Файл MappingStateTransitionException.cs

```
using System.Runtime.Serialization;

namespace Common.Exceptions
{
    /// <summary>
    /// Represents lambda configuration validation failure.
    /// </summary>
    [Serializable]
    public class MappingStateTransitionException : Exception
    {
        /// <summary>
        /// Initializes a new instance of <see
cref="MappingStateTransitionException"/> class.
        /// </summary>
        public MappingStateTransitionException()
        {
        }

        /// <summary>
        /// Initializes a new instance of <see
cref="MappingStateTransitionException"/> class.
        /// </summary>
        /// <param name="message">Exception message.</param>
        public MappingStateTransitionException(string? message) : base(message)
        {
        }

        /// <summary>
```

```
        /// Initializes a new instance of <see
cref="MappingStateTransitionException"/> class.
        /// </summary>
        /// <param name="message">Exception message.</param>
        /// <param name="innerException">Inner exception.</param>
        public MappingStateTransitionException(string? message, Exception?
innerException) : base(message, innerException)
        {
        }

        /// <summary>
        /// Initializes a new instance of <see
cref="MappingStateTransitionException"/> class.
        /// </summary>
        /// <param name="info"><see cref="SerializationInfo"/></param>
        /// <param name="context"><see cref="StreamingContext"/></param>
        protected MappingStateTransitionException(SerializationInfo info,
StreamingContext context) : base(info, context)
        {
        }
    }
}
```

Файл ProductNotFoundException.cs

```
using System.Runtime.Serialization;

namespace Common.Exceptions
{
    /// <summary>
    /// Represents an error during product retrieving operation.
    /// </summary>
    [Serializable]
    public class ProductNotFoundException : Exception
    {
        /// <summary>
        /// Initializes a new instance of <see cref="ProductNotFoundException"/>
class.
        /// </summary>
        public ProductNotFoundException() : this("Product is not found.")
        {
        }

        /// <summary>
        /// Initializes a new instance of <see cref="ProductNotFoundException"/>
class.
        /// </summary>
        /// <param name="message">Exception message.</param>
        public ProductNotFoundException(string? message) : base(message)
        {
        }

        /// <summary>
        /// Initializes a new instance of <see cref="ProductNotFoundException"/>
class.
        /// </summary>
        /// <param name="message">Exception message.</param>
        /// <param name="innerException">Inner exception.</param>
        public ProductNotFoundException(string? message, Exception? innerException)
: base(message, innerException)
        {
        }

        /// <summary>
```

```
        /// Initializes a new instance of <see cref="ProductNotFoundException"/>
class.
        /// </summary>
        /// <param name="info"><see cref="SerializationInfo"/></param>
        /// <param name="context"><see cref="StreamingContext"/></param>
        protected ProductNotFoundException(SerializationInfo info, StreamingContext
context) : base(info, context)
        {
        }
    }
}
```

Файл UserNotFoundException.cs

```
using System.Runtime.Serialization;

namespace Common.Exceptions
{
    /// <summary>
    /// Represents an error during user retrieving operation.
    /// </summary>
    [Serializable]
    public class UserNotFoundException : Exception
    {
        /// <summary>
        /// Initializes a new instance of <see cref="UserNotFoundException"/> class.
        /// </summary>
        public UserNotFoundException() : this("User is not found.")
        {
        }

        /// <summary>
        /// Initializes a new instance of <see cref="UserNotFoundException"/> class.
        /// </summary>
        /// <param name="message">Exception message.</param>
        public UserNotFoundException(string? message) : base(message)
        {
        }

        /// <summary>
        /// Initializes a new instance of <see cref="UserNotFoundException"/> class.
        /// </summary>
        /// <param name="message">Exception message.</param>
        /// <param name="innerException">Inner exception.</param>
        public UserNotFoundException(string? message, Exception? innerException) :
base(message, innerException)
        {
        }

        /// <summary>
        /// Initializes a new instance of <see cref="UserNotFoundException"/> class.
        /// </summary>
        /// <param name="info"><see cref="SerializationInfo"/></param>
        /// <param name="context"><see cref="StreamingContext"/></param>
        protected UserNotFoundException(SerializationInfo info, StreamingContext
context) : base(info, context)
        {
        }
    }
}
```

Файл DatabaseConfigurationExtensions.cs

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
```

```
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

namespace Common.Extensions
{
    /// <summary>
    /// DynamoDB configuration extension class.
    /// </summary>
    public static class DatabaseConfigurationExtensions
    {
        /// <summary>
        /// Configures dynamoDB services in the DI container.
        /// </summary>
        /// <param name="services"><see cref="IServiceCollection"/></param>
        /// <param name="configuration"><see cref="IConfiguration"/></param>
        /// <returns><see cref="IServiceCollection"/></returns>
        public static IServiceCollection ConfigureDynamoDB(this IServiceCollection
services, IConfiguration configuration)
        {
            return services
                .AddDefaultAWSOptions(configuration.GetAWSOptions())
                .AddAWSService<IAmazonDynamoDB>()
                .AddTransient<IDynamoDBContext, DynamoDBContext>();
        }
    }
}
```

Файл LoggerConfigurationExtensions.cs

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Serilog;
using Serilog.Events;

namespace Common.Extensions
{
    /// <summary>
    /// Logger configuration extension class.
    /// </summary>
    public static class LoggerConfigurationExtensions
    {
        /// <summary>
        /// Configure Serilog logger.
        /// </summary>
        /// <param name="serviceCollection"><see cref="IServiceCollection"/></param>
        /// <returns><see cref="ServiceCollection"/></returns>
        public static IServiceCollection ConfigureLogging(this IServiceCollection
serviceCollection, LogEventLevel logEventLevel = LogEventLevel.Debug)
        {
            var logger = new LoggerConfiguration()
                .Enrich.FromLogContext()
                .MinimumLevel.Debug()
                .WriteTo.Console()
                .CreateLogger();

            serviceCollection.AddLogging(builder =>
            {
                builder.ClearProviders();
                builder.AddSerilog(logger);
            });

            return serviceCollection;
        }
    }
}
```

Файл ParametersConfigurationExtensions.cs

```
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

namespace Common.Extensions
{
    /// <summary>
    /// Extension class to work with lambda environment variables.
    /// </summary>
    public static class ParametersConfigurationExtensions
    {
        /// <summary>
        /// Injects lambda parameters in the DI container.
        /// </summary>
        /// <typeparam name="TOptions">Type of options being configured.</typeparam>
        /// <param name="services"><see cref="IServiceCollection"/></param>
        /// <param name="configuration"><see cref="IConfiguration"/></param>
        /// <returns><see cref="IServiceCollection"/></returns>
        public static IServiceCollection ConfigureLambdaVariables<TOptions>(this
IServiceCollection services, IConfiguration configuration) where TOptions : class
        {
            services.Configure<TOptions>(configuration.GetSection("Parameters"));

            return services;
        }
    }
}
```

Файл SnsConfigurationExtensions.cs

```
using Amazon.SimpleNotificationService;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

namespace Common.Extensions
{
    /// <summary>
    /// Sns configuration extension class.
    /// </summary>
    public static class SnsConfigurationExtensions
    {
        /// <summary>
        /// Configures sns in the DI container.
        /// </summary>
        /// <param name="services"><see cref="IServiceCollection"/></param>
        /// <param name="configuration"><see cref="IConfiguration"/></param>
        /// <returns><see cref="IServiceCollection"/></returns>
        public static IServiceCollection ConfigureSns(this IServiceCollection
services)
        {
            return services
                .AddTransient<IAmazonSimpleNotificationService,
AmazonSimpleNotificationServiceClient>();
        }
    }
}
```

Файл SwaggerConfigrationExtensions.cs

```
using Common.Extensions;
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.OpenApi.Models;
using Swashbuckle.AspNetCore.SwaggerGen;
```

```
namespace Common.Extensions
{
    public static class SwaggerConfigrationExtensions
    {
        public static IServiceCollection ConfigureSwaggerServices(this
IServiceCollection services, OpenApiInfo swaggerInfo)
        {
            services.AddSwaggerGen(options =>
            {
                options.SwaggerDoc("v1", swaggerInfo);

                AddSwaggerXmlComments(options);
            });

            return services;
        }

        public static IApplicationBuilder UseSwagger(this IApplicationBuilder app,
string projectPrefix, IConfiguration configuration)
        {
            app.UseSwagger(c =>
            {
                c.RouteTemplate = $"{projectPrefix}/{{documentName}}/swagger.json";
            })
                .UseSwaggerUI(c =>
                {
                    c.SwaggerEndpoint("v1/swagger.json", "Users API implementation
for License Management Service.");
                    c.RoutePrefix = projectPrefix;
                });

            return app;
        }

        private static void AddSwaggerXmlComments(SwaggerGenOptions options)
        {
            foreach (var xmlDocFile in
Directory.EnumerateFiles(AppContext.BaseDirectory, "*.xml"))
            {
                options.IncludeXmlComments(xmlDocFile);
            }
        }
    }
}
```

Файл IEventBridgeSchedulerService.cs

```
namespace Common.Interfaces
{
    /// <summary>
    /// Interface for Event Bridge Scheduler service.
    /// </summary>
    public interface IEventBridgeSchedulerService
    {
        /// <summary>
        /// Sets Event Brdige rule state to DISABLED.
        /// </summary>
        /// <returns></returns>
        Task CancelCircuitClosureTrialAsync();

        /// <summary>
        /// Sets Event Bridge rule state to ENABLED.
        /// </summary>
        /// <returns></returns>
```

```
        Task ScheduleCircuitClosureTrialAsync(DateTimeOffset trialDate);
    }
}
```

Файл IEventSourceMappingClient.cs

```
namespace Common.Interfaces
{
    /// <summary>
    /// Interface of lambda event source mapping service.
    /// </summary>
    public interface IEventSourceMappingClient
    {
        /// <summary>
        /// Disables event source mapping for lambda.
        /// </summary>
        /// <returns></returns>
        public Task DisableEventSourceMappingAsync();

        /// <summary>
        /// Enables event source mapping for lambda.
        /// </summary>
        /// <returns></returns>
        public Task EnableEventSourceMappingAsync();
    }
}
```

Файл ILambdaContextAccessor.cs

```
using Amazon.Lambda.Core;

namespace Common.Interfaces
{
    /// <summary>
    /// Interface for lambda function context wrapper.
    /// </summary>
    public interface ILambdaContextAccessor
    {
        /// <summary>
        /// <see cref="ILambdaContext"/>
        /// </summary>
        ILambdaContext Context { get; }
    }
}
```

Файл IMessage.cs

```
using Common.Entities;

namespace Common.Interfaces
{
    /// <summary>
    /// Represents a message in SNS or SQS.
    /// </summary>
    public interface IMessage
    {
        /// <summary>
        /// Gets the Id of the entity message refers to.
        /// </summary>
        public string EntityId { get; }

        /// <summary>
        /// Gets the action being performed of the Entity.
        /// </summary>
        public ProcessAction Action { get; }
```

```
        /// <summary>
        /// Gets the body of the message.
        /// </summary>
        public object Content { get; }

        /// <summary>
        /// Gets or sets the datetime that the message was sent.
        /// </summary>
        public DateTimeOffset MessageSentOn { get; set; }
    }

    /// <summary>
    /// Represents a strongly typed instance of a message.
    /// </summary>
    /// <typeparam name="T">The type of the content.</typeparam>
    public interface IMessage<T> : IMessage
    {
        /// <summary>
        /// Gets or sets the message body.
        /// </summary>
        new T Content { get; set; }
    }
}
```

Файл IReadRepository.cs

```
namespace Common.Interfaces
{
    /// <summary>
    /// Interface of datastore read service.
    /// </summary>
    /// <typeparam name="TEntity">Data transfer object.</typeparam>
    public interface IReadRepository<TEntity>
    {
        /// <summary>
        /// Gets entity by id.
        /// </summary>
        /// <param name="id">Entity unique indentifier.</param>
        /// <returns>Entity dto.</returns>
        Task<TEntity> GetByIdAsync(Guid id);
    }
}
```

Файл ISnsClient.cs

```
namespace Common.Interfaces
{
    /// <summary>
    /// Interface of SNS message publisher service.
    /// </summary>
    public interface ISnsClient
    {
        /// <summary>
        /// Publishes a message to the bus.
        /// </summary>
        /// <typeparam name="T">Type of the message payload.</typeparam>
        /// <param name="topicArn">Target topic arn.</param>
        /// <param name="message">The message to be sent.</param>
        /// <returns></returns>
        Task PublishToTopicAsync<T>(string topicArn, IMessage<T> message);
    }
}
```

Файл ISqsClient.cs

```
using Amazon.SQS.Model;
using static Amazon.Lambda.SQSEvents.SQSEvent;
```

```
namespace Common.Interfaces
{
    /// <summary>
    /// Interface of service to work with AWS SQS queues.
    /// </summary>
    public interface ISqsClient
    {
        /// <summary>
        /// Adds message to the SQS queue.
        /// </summary>
        /// <param name="message"><see cref="SQSMessage"/></param>
        /// <param name="queueUrl">Queue url.</param>
        /// <returns>Task.</returns>
        Task EnqueueAsync(SQSMessage message, Uri queueUrl);

        /// <summary>
        /// Takes last message from the SQS queue.
        /// </summary>
        /// <param name="queueUrl">Queue url.</param>
        /// <returns><see cref="Message"/></returns>
        Task<Message?> DequeueAsync(Uri queueUrl);

        /// <summary>
        /// Deletes message from the SQS queue.
        /// </summary>
        /// <param name="receiptHandle">Receipt handle.</param>
        /// <param name="queueUrl">Queue url.</param>
        /// <returns>Task.</returns>
        Task DeleteMessageAsync(string receiptHandle, Uri queueUrl);
    }
}
```

Файл ISqsEventProcessingService.cs

```
using Newtonsoft.Json.Linq;

namespace Common.Interfaces
{
    /// <summary>
    /// Interface of SQS Event processor service.
    /// </summary>
    public interface ISqsEventProcessingService
    {
        /// <summary>
        /// Processes the SQS Event and logs incomming message.
        /// </summary>
        /// <param name="input">JSON input.</param>
        /// <returns>Task.</returns>
        Task ProcessAsync(JObject input);
    }
}
```

Файл IWriteRepository.cs

```
namespace Common.Interfaces
{
    /// <summary>
    /// Interface of datastore write service.
    /// </summary>
    /// <typeparam name="TEntity">Data transfer object.</typeparam>
    public interface IWriteRepository<TEntity>
    {
        /// <summary>
        /// Creates or updates entity in the datastore.
```

```
        /// </summary>
        /// <param name="entity">Entity to be created.</param>
        /// <returns>Saved user dto.</returns>
        Task<TEntity> SaveAsync(TEntity entity);

        /// <summary>
        /// Deletes entity from the datastore.
        /// </summary>
        /// <param name="id">Entity unique identifier.</param>
        /// <returns>Deleted entity dto.</returns>
        Task<TEntity> DeleteAsync(Guid id);
    }
}
```

Файл LicenseMapper.cs

```
using Common.Entities;

namespace Common.Mappers
{
    public static class LicenseMapper
    {
        public static LicenseDto MapToDto(this LicenseCreateModel licenseModel, Guid
productId)
        {
            var licenseDto = new LicenseDto
            {
                ProductId = productId,
                PriceAmount = licenseModel.PriceAmount,
                Currency = licenseModel.Currency
            };

            return licenseDto;
        }
    }
}
```

Файл UnhandledExceptionLoggingMiddleware.cs

```
using Common.Exceptions;
using FluentValidation;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;

namespace Common.Middleware
{
    /// <summary>
    /// Logs unhandled exceptions during request lifetime.
    /// </summary>
    public class UnhandledExceptionLoggingMiddleware
    {
        private readonly RequestDelegate _next;
        private readonly ILogger<UnhandledExceptionLoggingMiddleware> _logger;

        /// <summary>
        /// Initializes a new instance of <see
cref="UnhandledExceptionLoggingMiddleware"/> class.
        /// </summary>
        /// <param name="next"><see cref="RequestDelegate"/></param>
        /// <param name="logger">Logger instance.</param>
        /// <exception cref="ArgumentNullException"></exception>
        public UnhandledExceptionLoggingMiddleware(RequestDelegate next,
ILogger<UnhandledExceptionLoggingMiddleware> logger)
        {
            _next = next ?? throw new ArgumentNullException(nameof(next));
            _logger = logger;
```

```
        }

        /// <summary>
        /// Invokes http context and handles exceptions.
        /// </summary>
        /// <param name="context"><see cref="HttpContext"/></param>
        /// <returns>Task.</returns>
        /// <exception cref="ArgumentNullException">Context is null.</exception>
        public async Task InvokeAsync(HttpContext context)
        {
            if (context == null)
            {
                throw new ArgumentNullException(nameof(context));
            }

            try
            {
                await _next(context);
            }
            catch (ValidationException ex)
            {
                const string message = "Entity validation failed.";
                _logger.LogWarning(ex, message);
                context.Response.StatusCode = StatusCodes.Status400BadRequest;
            }
            catch (Exception ex) when (ex is UserNotFoundException ||
                                       ex is ProductNotFoundException ||
                                       ex is LicenseNotFoundException)
            {
                const string message = "Entity does not exist";
                _logger.LogWarning(ex, message);
                context.Response.StatusCode = StatusCodes.Status404NotFound;
            }
            catch (Exception ex)
            {
                const string message = "An unhandled exception occured during
request processing.";
                _logger.LogError(ex, message);
                context.Response.Clear();
                context.Response.StatusCode =
StatusCodes.Status500InternalServerError;
            }
        }
    }
}
```

Файл EventBridgeOptions.cs

```
using System.Diagnostics.CodeAnalysis;

namespace Common.Options
{
    /// <summary>
    /// Contains configuration parameters for Event Bridge Rule.
    /// </summary>
    [ExcludeFromCodeCoverage]
    public class EventBridgeOptions
    {
        /// <summary>
        /// Gets or sets the Event Bridge Rule name.
        /// </summary>
        public string RuleName { get; set; } = null!;
    }
}
```

Файл EventBridgeSchedulerService.cs

```
using Amazon.EventBridge;
using Amazon.EventBridge.Model;
using Common.Interfaces;
using Common.Options;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using System.Net;

namespace Common.Services
{
    /// <summary>
    /// Event Bridge Scheduler service.
    /// </summary>
    public class EventBridgeSchedulerService : IEventBridgeSchedulerService
    {
        private readonly IAmazonEventBridge _amazonEventBridge;
        private readonly EventBridgeOptions _eventBridgeOptions;
        private readonly ILogger<EventBridgeSchedulerService> _logger;

        /// <summary>
        /// Initializes a new instance of <see cref="EventBridgeSchedulerService"/>
class.
        /// </summary>
        /// <param name="amazonEventBridge"><see cref="IAmazonEventBridge"/></param>
        /// <param name="eventBridgeOptions"><see
cref="EventBridgeOptions"/></param>
        /// <param name="logger">Logger instance.</param>
        public EventBridgeSchedulerService(IAmazonEventBridge amazonEventBridge,
IOptions<EventBridgeOptions> eventBridgeOptions,
ILogger<EventBridgeSchedulerService> logger)
        {
            _amazonEventBridge = amazonEventBridge;
            _eventBridgeOptions = eventBridgeOptions.Value;
            _logger = logger;
        }

        /// <inheritdoc/>
        public Task CancelCircuitClosureTrialAsync()
        {
            return ChangeEventBridgeRuleStateAsync(RuleState.DISABLED, "rate(365
days)");
        }


        /// <inheritdoc/>
        public Task ScheduleCircuitClosureTrialAsync(DateTimeOffset trialDate)
        {
            return ChangeEventBridgeRuleStateAsync(RuleState.ENABLED,
GetCronExpression(trialDate));
        }

        private async Task ChangeEventBridgeRuleStateAsync(RuleState state, string
scheduleExpression)
        {
            _logger.LogDebug("Changing EventBridge {rule} rule to state {state}",
_eventBridgeOptions.RuleName, state);
            var request = new PutRuleRequest
            {
                Name = _eventBridgeOptions.RuleName,
                ScheduleExpression = scheduleExpression,
                State = state
            };
```

```
            PutRuleResponse? response = await
_amazonEventBridge.PutRuleAsync(request);
            if (response.HttpStatusCode != HttpStatusCode.OK)
            {
                throw new HttpRequestException($"Failed to change EventBridge
{_eventBridgeOptions.RuleName} rule to {state} state", null,
response.HttpStatusCode);
            }
            _logger.LogInformation("Changed EventBridge {rule} rule to {state}
state", _eventBridgeOptions.RuleName, state);
        }

        private static string GetCronExpression(DateTimeOffset trialDate)
        {
            return trialDate - DateTimeOffset.UtcNow <= TimeSpan.FromMinutes(1)
                ? "cron(0/1 * * * ? *)"
                : $"cron({trialDate.Minute} {trialDate.Hour} {trialDate.Day} * ?
*)";
        }
    }
}
```

Файл SnsClient.cs

```
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;
using Common.Interfaces;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;

namespace Common.Services
{
    /// <summary>
    /// Sns message publisher service.
    /// </summary>
    public class SnsClient : ISnsClient
    {
        private readonly IAmazonSimpleNotificationService
_simpleNotificationService;
        private readonly ILogger<SnsClient> _logger;

        /// <summary>
        /// Initializes a new instance of <see cref="SnsClient"/> class.
        /// </summary>
        /// <param name="simpleNotificationService"><see
cref="IAmazonSimpleNotificationService"/></param>
        /// <param name="logger">Logger instance.</param>
        public SnsClient(IAmazonSimpleNotificationService simpleNotificationService,
ILogger<SnsClient> logger)
        {
            _simpleNotificationService = simpleNotificationService;
            _logger = logger;
        }

        /// <inheritdoc/>
        public async Task PublishToTopicAsync<T>(string topicArn, IMessage<T>
message)
        {
            _logger.LogDebug("Started publishing the content: {@content} to the SNS
topic: {topicArn}", message, topicArn);

            var content = JsonConvert.SerializeObject(message, Formatting.None);
```

```
            _logger.LogDebug("Serialized content: {content}", content);

            var request = new PublishRequest
            {
                TopicArn = topicArn,
                Message = content
            };
            _logger.LogDebug("Created a new SNS publish request: {@request}",
request);

            await _simpleNotificationService.PublishAsync(topicArn, content);
            _logger.LogInformation("Successfully published message: {@content} to
the SNS topic: {topicArn}", content);
        }
    }
}
```

Файл SqsClient.cs

```
using Amazon.Lambda.SQSEvents;
using Amazon.SQS;
using Amazon.SQS.Model;
using Common.Interfaces;
using Microsoft.Extensions.Logging;

namespace Common.Services
{
    /// <summary>
    /// Service to work with Amazon SQS queues.
    /// </summary>
    public class SqsClient : ISqsClient
    {
        private readonly IAmazonSQS _sqs;
        private readonly ILogger<SqsClient> _logger;

        public SqsClient(IAmazonSQS sqs, ILogger<SqsClient> logger)
        {
            _sqs = sqs ?? throw new ArgumentNullException(nameof(sqs));
            _logger = logger ?? throw new ArgumentNullException(nameof(logger));
        }

        /// <inheritdoc/>
        public async Task EnqueueAsync(SQSEvent.SQSMessage message, Uri queueUrl)
        {
            if (message == null) throw new ArgumentNullException(nameof(message));

            await _sqs.SendMessageAsync(new SendMessageRequest { QueueUrl =
queueUrl.AbsoluteUri, MessageBody = message.Body });
            _logger.LogInformation("Message {messageId} added to {queueUrl}",
message.MessageId, queueUrl);
        }

        /// <inheritdoc/>
        public async Task<Message?> DequeueAsync(Uri queueUrl)
        {
            _logger.LogDebug("Trying to get message from the queue: {queueUrl}",
queueUrl);
            var receiveMessageRequest = new ReceiveMessageRequest
            {
                AttributeNames = { "All" },
                MaxNumberOfMessages = 1,
                QueueUrl = queueUrl.AbsoluteUri,
                VisibilityTimeout = 15,
```

```
                WaitTimeSeconds = 0
            };

            ReceiveMessageResponse receiveMessageResponse = await
_sqs.ReceiveMessageAsync(receiveMessageRequest);
            _logger.LogDebug("Received message response from the SQS queue:
{@response}", receiveMessageResponse);

            return receiveMessageResponse.Messages.FirstOrDefault();
        }

        /// <inheritdoc/>
        public async Task DeleteMessageAsync(string receiptHandle, Uri queueUrl)
        {
            await _sqs.DeleteMessageAsync(new DeleteMessageRequest { QueueUrl =
queueUrl.AbsoluteUri, ReceiptHandle = receiptHandle });
            _logger.LogInformation("Message {receiptHandle} deleted from the
{queueUrl}", receiptHandle, queueUrl);
        }
    }
}
```

Файл LambdaContextAccessor.cs

```
using Amazon.Lambda.Core;
using Common.Interfaces;

namespace Common.Utils
{
    /// <summary>
    /// Lambda context wrapper.
    /// </summary>
    public class LambdaContextAccessor : ILambdaContextAccessor
    {
        private static readonly AsyncLocal<ILambdaContext> AsyncStore = new ();

        /// <inheritdoc/>
        public ILambdaContext Context
        {
            get => AsyncStore.Value;
            set => AsyncStore.Value = value;
        }
    }
}
```

Файл CircuitBreakerPolicy.cs

```
namespace Resiliency.CircuitBreaker
{
    public class CircuitBreakerPolicy
    {
        private Func<Exception, bool> _exceptionFilter = _ => true;
        public CircuitState State { get; private set; }

        public CircuitBreakerPolicy(CircuitState currentState)
        {
            State = currentState ?? throw new
ArgumentNullException(nameof(currentState));
        }

        public CircuitBreakerPolicy Handle(Func<Exception, bool> exceptionFilter)
```

```
        {
            _exceptionFilter = exceptionFilter ?? throw new
ArgumentNullException(nameof(exceptionFilter));
            return this;
        }

        public Task ExecuteAsync(Func<Task> action)
        {
            if (action == null)
            {
                throw new ArgumentNullException(nameof(action));
            }

            return InnerExecuteAsync(action);
        }

        public Task ExecuteAsync<TResult>(Func<Task<TResult>> action)
        {
            if (action == null)
            {
                throw new ArgumentNullException(nameof(action));
            }

            return InnerExecuteAsync(action);
        }

        public void ClosePermanently()
        {
            State = new PermanentlyClosedState();
        }

        private async Task InnerExecuteAsync(Func<Task> action)
        {
            try
            {
                State = await State.ExecuteAsync(action);
            }
            catch (Exception exception) when (_exceptionFilter(exception) &&
exception is not OpenCircuitException)
            {
                State = State.RecordError();
                throw;
            }
        }

        private async Task<TResult> InnerExecuteAsync<TResult>(Func<Task<TResult>>
action)
        {
            try
            {
                ExecutionResult<TResult> executionResult = await
State.ExecuteAsync(action);
                State = executionResult.State;
                return executionResult.Result;
            }
            catch (Exception exception) when (_exceptionFilter(exception) &&
exception is not OpenCircuitException)
            {
                State = State.RecordError();
                throw;
            }
        }
    }
}
```

Файл CircuitState.cs

```
namespace Resiliency.CircuitBreaker
{
    public abstract class CircuitState
    {
        public abstract void Accept(ICircuitStateVisitor visitor);

        internal abstract CircuitState RecordError();
        internal abstract Task<CircuitState> ExecuteAsync(Func<Task> action);
        internal abstract Task<ExecutionResult<TResult>>
ExecuteAsync<TResult>(Func<Task<TResult>> action);
    }
}
```

Файл ClosedState.cs

```
using System.ComponentModel.DataAnnotations;

namespace Resiliency.CircuitBreaker
{
    public class ClosedState : CircuitState
    {
        private readonly OpenStateConfig _openStateConfig;
        private readonly ClosedStateConfig _closedStateConfig;

        public int ErrorCounter { get; private set; }
        public DateTimeOffset StartedAt { get; private set; }

        public ClosedState(OpenStateConfig openStateConfig, ClosedStateConfig
closedStateConfig) :
            this(0, DateTimeOffset.UtcNow, openStateConfig, closedStateConfig)
        { }

        public ClosedState([Range(0, int.MaxValue)] int errorCounter, DateTimeOffset
startedAt, OpenStateConfig openStateConfig, ClosedStateConfig closedStateConfig)
        {
            if (errorCounter < 0)
            {
                throw new ArgumentOutOfRangeException(nameof(errorCounter),
errorCounter, "Error counter must be a non-negative integer.");
            }
            ErrorCounter = errorCounter;

            if (startedAt > DateTimeOffset.UtcNow)
            {
                throw new ArgumentOutOfRangeException(nameof(startedAt), startedAt,
"Sampling start date must be in the past.");
            }
            StartedAt = startedAt;

            _openStateConfig = openStateConfig ?? throw new
ArgumentNullException(nameof(openStateConfig));
            _closedStateConfig = closedStateConfig ?? throw new
ArgumentNullException(nameof(closedStateConfig));
        }

        public override void Accept(ICircuitStateVisitor visitor)
        {
            if (visitor == null)
            {
                throw new ArgumentNullException(nameof(visitor));
            }

            visitor.Visit(this);
```

```
        }

        internal override CircuitState RecordError()
        {
            if (DateTimeOffset.UtcNow - StartedAt <
_closedStateConfig.SamplingDuration)
            {
                if(++ErrorCounter > _closedStateConfig.ErrorThreshold)
                {
                    return new OpenState(_openStateConfig, _closedStateConfig);
                }
            }
            else
            {
                ErrorCounter = 1;
                StartedAt = DateTimeOffset.UtcNow;
            }

            return this;
        }

        internal override async Task<CircuitState> ExecuteAsync(Func<Task> action)
        {
            await action();

            return this;
        }

        internal override async Task<ExecutionResult<TResult>>
ExecuteAsync<TResult>(Func<Task<TResult>> action)
        {
            TResult result = await action();

            return new ExecutionResult<TResult>(this, result);
        }
    }
}
```

Файл ClosedStateConfig.cs

```
using System.ComponentModel.DataAnnotations;

namespace Resiliency.CircuitBreaker
{
    public class ClosedStateConfig
    {
        public int ErrorThreshold { get; }
        public TimeSpan SamplingDuration { get; }

        public ClosedStateConfig([Range(0, int.MaxValue)] int errorThreshold,
TimeSpan samplingDuration)
        {
            if (errorThreshold < 0)
            {
                throw new ArgumentOutOfRangeException(nameof(errorThreshold),
errorThreshold, "Error threshold must be a non-negative integer.");
            }
            ErrorThreshold = errorThreshold;

            if (samplingDuration <= TimeSpan.Zero)
            {
                throw new ArgumentOutOfRangeException(nameof(samplingDuration),
samplingDuration, "Sampling duration must be greater than zero.");
            }
            SamplingDuration = samplingDuration;
```

```
        }
    }
}
```

Файл ExecutionResult.cs

```
namespace Resiliency.CircuitBreaker
{
    public class ExecutionResult<TResult>
    {
        public CircuitState State { get; }
        public TResult Result { get; }

        public ExecutionResult(CircuitState state, TResult result)
        {
            State = state ?? throw new ArgumentNullException(nameof(state));
            Result = result;
        }
    }
}
```

Файл HalfOpenState.cs

```
namespace Resiliency.CircuitBreaker
{
    public class HalfOpenState : CircuitState
    {
        private readonly OpenStateConfig _openStateConfig;
        private readonly ClosedStateConfig _closedStateConfig;

        public HalfOpenState(OpenStateConfig openStateConfig, ClosedStateConfig
closedStateConfig)
        {
            _openStateConfig = openStateConfig ?? throw new
ArgumentNullException(nameof(openStateConfig));
            _closedStateConfig = closedStateConfig ?? throw new
ArgumentNullException(nameof(closedStateConfig));
        }

        public override void Accept(ICircuitStateVisitor visitor)
        {
            if (visitor == null)
            {
                throw new ArgumentNullException(nameof(visitor));
            }

            visitor.Visit(this);
        }

        internal override CircuitState RecordError()
        {
            return new OpenState(_openStateConfig, _closedStateConfig);
        }

        internal override async Task<CircuitState> ExecuteAsync(Func<Task> action)
        {
            await action();

            return new ClosedState(_openStateConfig, _closedStateConfig);
        }

        internal override async Task<ExecutionResult<TResult>>
ExecuteAsync<TResult>(Func<Task<TResult>> action)
        {
            TResult result = await action();
```

```
            return new ExecutionResult<TResult>(new ClosedState(_openStateConfig,
_closedStateConfig), result);
        }
    }
}
```

Файл ICircuitStateVisitor.cs

```
namespace Resiliency.CircuitBreaker
{
    public interface ICircuitStateVisitor
    {
        void Visit(OpenState state);
        void Visit(ClosedState state);
        void Visit(HalfOpenState state);
        void Visit(PermanentlyClosedState state);
    }
}
```

Файл OpenCircuitException.cs

```
using System.Runtime.Serialization;

namespace Resiliency.CircuitBreaker
{
    /// <summary>
    /// Signals the upstream code that circuit is open and execution isn't allowed.
    /// </summary>
    [Serializable]
    public class OpenCircuitException : Exception
    {
        private const int DefaultBreakDurationInMinutes = 5;

        /// <summary>
        /// Default is 5 minutes from <see cref="DateTimeOffset.UtcNow"/>
        /// </summary>
        public DateTimeOffset OpenUntil { get; } =
DateTimeOffset.UtcNow.AddMinutes(DefaultBreakDurationInMinutes);

        public OpenCircuitException(DateTimeOffset openUntil)
        {
            OpenUntil = openUntil;
        }

        public OpenCircuitException() : base("Circuit is open.") { }

        public OpenCircuitException(Exception inner) : base("Circuit is open.",
inner) { }

        protected OpenCircuitException(SerializationInfo info, StreamingContext
context) : base(info, context)
        {
            if (info == null)
            {
                throw new ArgumentNullException("info");
            }

            OpenUntil = info.GetDateTime("OpenUntil");
        }

        public override void GetObjectData(SerializationInfo info, StreamingContext
context)
        {
            if (info == null)
```

```
            {
                throw new ArgumentNullException(nameof(info));
            }

            base.GetObjectData(info, context);
            info.AddValue("OpenUntil", OpenUntil.UtcDateTime, typeof(DateTime));
        }
    }
}
```

Файл OpenState.cs

```
namespace Resiliency.CircuitBreaker
{
    public class OpenState : CircuitState
    {
        private readonly OpenStateConfig _openStateConfig;
        private readonly ClosedStateConfig _closedStateConfig;

        public DateTimeOffset OpenedAt { get; }

        public OpenState(OpenStateConfig openStateConfig, ClosedStateConfig
closedStateConfig) : this(DateTimeOffset.UtcNow, openStateConfig, closedStateConfig)
        { }

        public OpenState(DateTimeOffset openedAt, OpenStateConfig openStateConfig,
ClosedStateConfig closedStateConfig)
        {
            if (openedAt > DateTimeOffset.UtcNow)
            {
                throw new ArgumentOutOfRangeException(nameof(openedAt), openedAt,
"Opening date must be in the past.");
            }

            OpenedAt = openedAt;
            _openStateConfig = openStateConfig ?? throw new
ArgumentNullException(nameof(openStateConfig));
            _closedStateConfig = closedStateConfig ?? throw new
ArgumentNullException(nameof(closedStateConfig));
        }

        public override void Accept(ICircuitStateVisitor visitor)
        {
            if (visitor == null)
            {
                throw new ArgumentNullException(nameof(visitor));
            }

            visitor.Visit(this);
        }

        internal override async Task<CircuitState> ExecuteAsync(Func<Task> action)
        {
            CircuitState state = EvaluateNextState();
            await action();
            return state;
        }

        internal override async Task<ExecutionResult<TResult>>
ExecuteAsync<TResult>(Func<Task<TResult>> action)
        {
            CircuitState state = EvaluateNextState();
            TResult result = await action();
            return new ExecutionResult<TResult>(state, result);
        }
```

```
        internal override CircuitState RecordError()
        {
            return this;
        }

        private CircuitState EvaluateNextState()
        {
            DateTimeOffset openUntil = OpenedAt.Add(_openStateConfig.Timeout);
            if (openUntil <= DateTimeOffset.UtcNow)
            {
                return new HalfOpenState(_openStateConfig, _closedStateConfig);
            }

            throw new OpenCircuitException(openUntil);
        }
    }
}
```

Файл OpenStateConfig.cs

```
namespace Resiliency.CircuitBreaker
{
    public class OpenStateConfig
    {
        private const int DefaultUpperLimitInMinutes = 30;

        /// <summary>
        /// The time frame circuit stays open until next trial.
        /// </summary>
        public TimeSpan Timeout { get; }

        public OpenStateConfig(TimeSpan timeout)
        {
            TimeSpan upperLimit = TimeSpan.FromMinutes(DefaultUpperLimitInMinutes);
            if (timeout <= TimeSpan.Zero || timeout > upperLimit)
            {
                throw new ArgumentOutOfRangeException(nameof(timeout), timeout,
$"Timeout must be positive and less than {upperLimit}.");
            }
            Timeout = timeout;
        }
    }
}
```

Файл PermanentlyClosedState.cs

```
namespace Resiliency.CircuitBreaker
{
    public class PermanentlyClosedState : CircuitState
    {
        public DateTimeOffset StartedAt { get; private set; }

        public PermanentlyClosedState() : this(DateTimeOffset.UtcNow) { }

        public PermanentlyClosedState(DateTimeOffset startedAt)
        {
            if (startedAt > DateTimeOffset.UtcNow)
            {
                throw new ArgumentOutOfRangeException(nameof(startedAt), startedAt,
"Starting date must be in the past.");
            }

            StartedAt = startedAt;
```

```
        }

        public override void Accept(ICircuitStateVisitor visitor)
        {
            if (visitor == null)
            {
                throw new ArgumentNullException(nameof(visitor));
            }

            visitor.Visit(this);
        }

        internal override CircuitState RecordError()
        {
            return this;
        }

        internal override async Task<CircuitState> ExecuteAsync(Func<Task> action)
        {
            await action();

            return this;
        }

        internal override async Task<ExecutionResult<TResult>>
ExecuteAsync<TResult>(Func<Task<TResult>> action)
        {
            TResult result = await action();

            return new ExecutionResult<TResult>(this, result);
        }
    }
}
```

Файл ServiceProviderBuilder.cs

```
using Common.Extensions;
using Common.Interfaces;
using LicenseManagementLambda.Interfaces;
using LicenseManagementLambda.Options;
using LicenseManagementLambda.Repositories;
using LicenseManagementLambda.Services;
using System.Diagnostics.CodeAnalysis;

namespace LicenseManagementLambda.Builders
{
    /// <summary>
    /// Provides a creation service of <see cref="IServiceProvider"/> instance.
    /// </summary>
    [ExcludeFromCodeCoverage]
    public class ServiceProviderBuilder
    {
        private readonly IConfiguration _configuration;

        public ServiceProviderBuilder()
        {
            _configuration = new ConfigurationBuilder()
                .SetBasePath(Directory.GetCurrentDirectory())
                .AddJsonFile("appsettings.json", optional: false, reloadOnChange:
true)
                .AddEnvironmentVariables()
                .Build();
        }

        /// <summary>
```

```
        /// Creates a <see cref="IServiceProvider"/> containing services provided
for solution.
        /// </summary>
        /// <param name="serviceCollection"></param>
        /// <returns><see cref="ServiceProvider"/></returns>
        /// <exception cref="ArgumentNullException"></exception>
        public IServiceProvider Build(IServiceCollection serviceCollection)
        {
            if (serviceCollection == null)
            {
                throw new ArgumentNullException(nameof(serviceCollection));
            }


serviceCollection.ConfigureLambdaVariables<LambdaParameters>(_configuration);
            serviceCollection.ConfigureLogging();
            serviceCollection.ConfigureDynamoDB(_configuration);

            AddServices(serviceCollection);

            serviceCollection.AddHttpClient("ProductsAPI", httpClient =>
            {
                httpClient.BaseAddress = new
Uri(_configuration.GetSection("Parameters:ProductsApiUrl").Value);
                httpClient.DefaultRequestHeaders.Add("Accept", "application/json");
            });
            serviceCollection.AddHttpClient("UsersAPI", httpClient =>
            {
                httpClient.BaseAddress = new
Uri(_configuration.GetSection("Parameters:UsersApiUrl").Value);
                httpClient.DefaultRequestHeaders.Add("Accept", "application/json");
            });

            var serviceProvider = serviceCollection.BuildServiceProvider();

            return serviceProvider;
        }

        private static void AddServices(IServiceCollection services)
        {
            services.AddScoped<ILicenseManagementService,
LicenseManagementService>();
            services.AddScoped<ILicenseRepository, LicenseRepository>();
            services.AddScoped<IProductEntitlementManagementService,
ProductEntitlementManagementService>();
            services.AddScoped<IProductEntitlementRepository,
ProductEntitlementRepository>();
            services.AddScoped<ISqsEventProcessingService,
SqsEventProcessingService>();
        }
    }
}
```

Файл LicenseController.cs

```
using Common.Entities;
using LicenseManagementLambda.Interfaces;
using Microsoft.AspNetCore.Mvc;
using System.ComponentModel.DataAnnotations;

namespace LicenseManagementLambda.Controllers;

/// <summary>
```

```
/// API controller for licenses management in License Management Service.
/// </summary>
[ApiController]
[Route("license-api/licenses")]
[Produces("application/json")]
public class LicenseController : ControllerBase
{
    private readonly ILicenseManagementService _licenseManagementService;
    private readonly ILogger<LicenseController> _logger;

    /// <summary>
    /// Initializes a new instance of <see cref="LicenseController"/> class.
    /// </summary>
    /// <param name="licenseManagementService"><see
cref="ILicenseManagementService"/></param>
    /// <param name="logger">Logger instance.</param>
    public LicenseController(ILicenseManagementService licenseManagementService,
ILogger<LicenseController> logger)
    {
        _licenseManagementService = licenseManagementService;
        _logger = logger;
    }

    /// <summary>
    /// Gets license by ID.
    /// </summary>
    /// <param name="licenseId">License unique identifier.</param>
    /// <returns>License entity.</returns>
    /// <remarks>
    /// Example url call:
    ///
    /// GET <code>license-management/license-api/licenses?licenseId=ebff8ad4-24f9-
4be7-a15d-529f64ede7c6</code>
    /// </remarks>
    [HttpGet]
    public async Task<IActionResult> GetLicense([Required, FromQuery] Guid
licenseId)
    {
        var license = await
_licenseManagementService.GetLicenseByIdAsync(licenseId);

        return Ok(license);
    }

    /// <summary>
    /// Creates new license.
    /// </summary>
    /// <param name="productId">Product unique identifier.</param>
    /// <param name="licenseModel"><see cref="LicenseCreateModel"/></param>
    /// <returns>Created license definition.</returns>
    /// <remarks>
    /// Example url call:
    ///
    /// POST <code>license-management/license-api/licences?productId=ebff8ad4-24f9-
4be7-a15d-529f64ede7c6</code>
    /// </remarks>
    [HttpPost]
    public async Task<IActionResult> CreateLicense([FromQuery, Required] Guid
productId, [FromBody, Required]LicenseCreateModel licenseModel)
    {
        var license = await _licenseManagementService.CreateLicenseAsync(productId,
licenseModel);

        return CreatedAtAction(nameof(CreateLicense), license);
```

```
    }

    /// <summary>
    /// Updates an existing license.
    /// </summary>
    /// <param name="licenseDto"><see cref="LicenseDto"/></param>
    /// <returns>Updated license definition.</returns>
    /// <remarks>
    /// Example url call:
    ///
    /// PUT <code>license-management/license-api/licences</code>
    /// </remarks>
    [HttpPut]
    public async Task<IActionResult> UpdateLicense([FromBody, Required] LicenseDto
licenseDto)
    {
        var license = await
_licenseManagementService.UpdateLicenseAsync(licenseDto);

        return Ok(license);
    }

    /// <summary>
    /// Deletes an existing license.
    /// </summary>
    /// <param name="licenseId">License unique identifier.</param>
    /// <returns>Deleted license definition.</returns>
    /// <remarks>
    /// Example url call:
    ///
    /// DELETE <code>license-management/products-api/products?licenseId=ebff8ad4-
24f9-4be7-a15d-529f64ede7c6</code>
    /// </remarks>
    [HttpDelete]
    public async Task<IActionResult> DeleteLicense([Required, FromQuery] Guid
licenseId)
    {
        var license = await _licenseManagementService.DeleteLicenseAsync(licenseId);

        return Ok(license);
    }
}
```

Файл ProductEntitlementController.cs

```
using LicenseManagementLambda.Interfaces;
using Microsoft.AspNetCore.Mvc;
using System.ComponentModel.DataAnnotations;

namespace LicenseManagementLambda.Controllers;

/// <summary>
/// API controller for product entitlement management in License Management Service.
/// </summary>
[ApiController]
[Route("license-api/entitlements")]
[Produces("application/json")]
public class ProductEntitlementController : ControllerBase
{
    private readonly IProductEntitlementManagementService
_productEntitlementManagementService;
    private readonly ILogger<ProductEntitlementController> _logger;

    public ProductEntitlementController(IProductEntitlementManagementService
productEntitlementManagementService, ILogger<ProductEntitlementController> logger)
```

```
    {
        _productEntitlementManagementService = productEntitlementManagementService;
        _logger = logger;
    }

    /// <summary>
    /// Gets product entitlement by ID.
    /// </summary>
    /// <param name="entitlementId">Product's entitlement unique identifier.</param>
    /// <returns>Product entitlement entity.</returns>
    /// <remarks>
    /// Example url call:
    ///
    /// GET <code>license-management/license-
api/entitlements?entitlementId=ebff8ad4-24f9-4be7-a15d-529f64ede7c6</code>
    /// </remarks>
    [HttpGet("{id}")]
    public async Task<IActionResult> GetEntitlement([Required, FromQuery] Guid
entitlementId)
    {
        var entitlement = await
_productEntitlementManagementService.GetEntitlementByIdAsync(entitlementId);

        return Ok(entitlement);
    }

    /// <summary>
    /// Creates new product entitlement.
    /// </summary>
    /// <param name="licenseId">License unique identifier.</param>
    /// <param name="userId">User's unique identifier.</param>
    /// <returns>Created product entitlement definition.</returns>
    /// <remarks>
    /// Example url call:
    ///
    /// POST <code>license-management/license-
api/entitlements?licenseId={id}&amp;userId={id}</code>
    /// </remarks>
    [HttpPost]
    public async Task<IActionResult> CreateEntitlement([FromQuery, Required] Guid
licenseId, [FromQuery, Required] Guid userId)
    {
        var license = await
_productEntitlementManagementService.CreateEntitlementAsync(licenseId, userId);

        return CreatedAtAction(nameof(CreateEntitlement), license);
    }
}
```

Файл ILicenseManagementService.cs

```
using Common.Entities;

namespace LicenseManagementLambda.Interfaces
{
    /// <summary>
    /// Interface of license CRUD management service.
    /// </summary>
    public interface ILicenseManagementService
    {
        /// <summary>
        /// Gets license from the datastore.
        /// </summary>
        /// <param name="licenseId">License's unqiue indentifier.</param>
        /// <returns>License entity.</returns>
```

```
        Task<LicenseDto> GetLicenseByIdAsync(Guid licenseId);

        /// <summary>
        /// Manages license creation operation.
        /// </summary>
        /// <param name="productId">Product's unqiue indentifier.</param>
        /// <param name="licenseModel"><see cref="LicenseCreateModel"/></param>
        /// <returns>Created entity.</returns>
        Task<LicenseDto> CreateLicenseAsync(Guid productId, LicenseCreateModel
licenseModel);

        /// <summary>
        /// Manages license deletion operation.
        /// </summary>
        /// <param name="licenseId"><see cref="LicenseDto"/></param>
        /// <returns>Deleted entity.</returns>
        Task<LicenseDto> DeleteLicenseAsync(Guid licenseId);

        /// <summary>
        /// Manages license update operation.
        /// </summary>
        /// <param name="licenseDto"><see cref="LicenseDto"/></param>
        /// <returns>Updated entity.</returns>
        Task<LicenseDto> UpdateLicenseAsync(LicenseDto licenseDto);
    }
}
```

Файл IProductEntitlementManagementService.cs

```
using Common.Entities;

namespace LicenseManagementLambda.Interfaces
{
    public interface IProductEntitlementManagementService
    {
        /// <summary>
        /// Gets entitlement from the datastore.
        /// </summary>
        /// <param name="entitlementId">Entitlement's unqiue indentifier.</param>
        /// <returns>Product entitlement entity.</returns>
        Task<ProductEntitlementDto> GetEntitlementByIdAsync(Guid entitlementId);

        /// <summary>
        /// Manages entitlement creation operation.
        /// </summary>
        /// <param name="licenseId">License's unqiue indentifier.</param>
        /// <param name="userId">Users's unqiue indentifier.</param>
        /// <returns>Created entity.</returns>
        Task<ProductEntitlementDto> CreateEntitlementAsync(Guid licenseId, Guid
userId);

        /// <summary>
        /// Manages entitlement deletion operation.
        /// </summary>
        /// <param name="entitlementId">Entitlement's unqiue indentifier.</param>
        /// <returns>Deleted entity.</returns>
        Task<ProductEntitlementDto> DeleteEntitlementAsync(Guid entitlementId);

        /// <summary>
        /// Manages entitlement update operation.
        /// </summary>
        /// <param name="entitlementDto"><see cref="ProductEntitlementDto"/></param>
        /// <returns>Updated entity.</returns>
```

```
        Task<ProductEntitlementDto> UpdateEntitlementAsync(ProductEntitlementDto
entitlementDto);

        Task UpdateUserDetails(BaseMessage<UserDto> details);

        Task UpdateProductDetails(BaseMessage<ProductDto> details);
    }
}
```

Файл IProductEntitlementRepository.cs

```
using Common.Entities;
using Common.Interfaces;

namespace LicenseManagementLambda.Interfaces
{
    /// <summary>
    /// Interface of product entitlement datastore.
    /// </summary>
    public interface IProductEntitlementRepository :
IReadRepository<ProductEntitlementDto>, IWriteRepository<ProductEntitlementDto>
    {
        /// <summary>
        /// Return entitlements corresponding to user.
        /// </summary>
        /// <param name="userId">User's unique identifier.</param>
        /// <returns><see cref="ProductEntitlementDto"/></returns>

        Task<IList<ProductEntitlementDto>> GetByUserIdAsync(Guid userId);

        /// <summary>
        /// Return entitlements corresponding to product.
        /// </summary>
        /// <param name="productId">Product's unique identifier.</param>
        /// <returns><see cref="ProductEntitlementDto"/></returns>
        Task<IList<ProductEntitlementDto>> GetByProductIdAsync(Guid productId);
    }
}
```

Файл LambdaParameters.cs

```
namespace LicenseManagementLambda.Options
{
    /// <summary>
    /// Contains lambda parameters.
    /// </summary>
    public class LambdaParameters
    {
        /// <summary>
        /// Products API url.
        /// </summary>
        public string ProductsApiUrl { get; set; }

        /// <summary>
        /// Users API url.
        /// </summary>
        public string UsersApiUrl { get; set; }
    }
}
```

Файл LicenseRepository.cs

```
using Amazon.DynamoDBv2.DataModel;
using Common.Entities;
using Common.Exceptions;
using LicenseManagementLambda.Interfaces;

namespace LicenseManagementLambda.Repositories
{
    /// <summary>
    /// Licenses datastore service.
    /// </summary>
    public class LicenseRepository : ILicenseRepository
    {
        private readonly IDynamoDBContext _dynamoDbContext;
        private readonly ILogger<LicenseRepository> _logger;

        /// <summary>
        /// Initializes a new instance of <see cref="LicenseRepository"/> class.
        /// </summary>
        /// <param name="dynamoDbContext"><see cref="IDynamoDBContext"/></param>
        /// <param name="logger">Logger instance.</param>
        public LicenseRepository(IDynamoDBContext dynamoDbContext,
ILogger<LicenseRepository> logger)
        {
            _dynamoDbContext = dynamoDbContext;
            _logger = logger;
        }

        /// <inheritdoc/>
        public async Task<LicenseDto> GetByIdAsync(Guid id)
        {
            _logger.LogDebug("Trying to get license entity with id: {id}", id);

            LicenseDto license = await _dynamoDbContext.LoadAsync<LicenseDto>(id);

            _logger.LogInformation("Successfully retrieved license entity:
{@entity}", license);

            return license;
        }

        /// <inheritdoc/>
        public async Task<LicenseDto> DeleteAsync(Guid id)
        {
            _logger.LogDebug("Trying to delete license entity with id: {id}", id);

            var config = new DynamoDBOperationConfig
            {
                IgnoreNullValues = false
            };

            LicenseDto license = await _dynamoDbContext.LoadAsync<LicenseDto>(id,
config);
            if (license is null)
            {
                throw new LicenseNotFoundException();
            }

            await _dynamoDbContext.DeleteAsync<LicenseDto>(id);
            _logger.LogInformation("Successfully deleted license entity:
{@entity}}", license);

            return license;
        }
```

```
        /// <inheritdoc/>
        public async Task<LicenseDto> SaveAsync(LicenseDto license)
        {
            _logger.LogDebug("Trying to save license entity: {@entity}", license);

            var config = new DynamoDBOperationConfig
            {
                IgnoreNullValues = false
            };

            await _dynamoDbContext.SaveAsync(license, config);
            _logger.LogInformation("Successfully save a user entity: {@entity}",
license);

            return license;
        }
    }
}
```

Файл ProductEntitlementRepository.cs

```
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Common.Entities;
using Common.Exceptions;
using LicenseManagementLambda.Interfaces;

namespace LicenseManagementLambda.Repositories
{
    /// <summary>
    /// Product entitlement datastore service.
    /// </summary>
    public class ProductEntitlementRepository : IProductEntitlementRepository
    {
        private readonly IDynamoDBContext _dynamoDbContext;
        private readonly ILogger<ProductEntitlementRepository> _logger;

        /// <summary>
        /// Initializes a new instance of <see cref="ProductEntitlementRepository"/>
class.
        /// </summary>
        /// <param name="dynamoDbContext"><see cref="IDynamoDBContext"/></param>
        /// <param name="logger">Logger instance.</param>
        public ProductEntitlementRepository(IDynamoDBContext dynamoDbContext,
ILogger<ProductEntitlementRepository> logger)
        {
            _dynamoDbContext = dynamoDbContext;
            _logger = logger;
        }

        /// <inheritdoc/>
        public async Task<ProductEntitlementDto> GetByIdAsync(Guid id)
        {
            _logger.LogDebug("Trying to get product entitlement entity with id:
{id}", id);

            ProductEntitlementDto entitlement = await
_dynamoDbContext.LoadAsync<ProductEntitlementDto>(id);

            _logger.LogInformation("Successfully retrieved product entitlement
entity: {@entity}", entitlement);

            return entitlement;
```

```
        }

        /// <inheritdoc/>
        public async Task<ProductEntitlementDto> DeleteAsync(Guid id)
        {
            _logger.LogDebug("Trying to delete product entitlement entity with id:
{id}", id);

            var config = new DynamoDBOperationConfig
            {
                IgnoreNullValues = false
            };

            ProductEntitlementDto entitlement = await
_dynamoDbContext.LoadAsync<ProductEntitlementDto>(id, config);
            if (entitlement is null)
            {
                throw new EntitlementNotFoundException();
            }

            await _dynamoDbContext.DeleteAsync<ProductEntitlementDto>(id);
            _logger.LogInformation("Successfully deleted product entitlement entity:
{@entity}}", entitlement);

            return entitlement;
        }

        /// <inheritdoc/>
        public async Task<ProductEntitlementDto> SaveAsync(ProductEntitlementDto
entitlement)
        {
            _logger.LogDebug("Trying to save product entitlement entity: {@entity}",
entitlement);

            var config = new DynamoDBOperationConfig
            {
                IgnoreNullValues = false
            };

            await _dynamoDbContext.SaveAsync(entitlement, config);
            _logger.LogInformation("Successfully saved a new product entitlement
entity: {@entity}", entitlement);

            return entitlement;
        }

        /// <inheritdoc/>
        public async Task<IList<ProductEntitlementDto>> GetByUserIdAsync(Guid
userId)
        {
            _logger.LogDebug("Trying to get product entitlement entity by user ID:
{user}", userId);

            List<ScanCondition> scanConditions = new() { new ScanCondition("UserId",
ScanOperator.Equal, userId) };
            var scanResult =
_dynamoDbContext.ScanAsync<ProductEntitlementDto>(scanConditions);
            _logger.LogDebug("Scan Result: {result}", scanResult);

            var searchResult = await scanResult.GetRemainingAsync();

            _logger.LogInformation("Retrieved product entitlements: {searchResult}",
searchResult);
```

```
            if (searchResult is null || !searchResult.Any())
            {
                throw new EntitlementNotFoundException();
            }

            return searchResult;
        }

        /// <inheritdoc/>
        public async Task<IList<ProductEntitlementDto>> GetByProductIdAsync(Guid
productId)
        {
            _logger.LogDebug("Trying to get product entitlement entity by product
ID: {product}", productId);

            List<ScanCondition> scanConditions = new() { new
ScanCondition("ProductId", ScanOperator.Equal, productId) };
            var scanResult =
_dynamoDbContext.ScanAsync<ProductEntitlementDto>(scanConditions);
            _logger.LogDebug("Scan Result: {result}", scanResult);

            var searchResult = await scanResult.GetRemainingAsync();

            _logger.LogInformation("Retrieved product entitlements: {searchResult}",
searchResult);

            if (searchResult is null || !searchResult.Any())
            {
                throw new EntitlementNotFoundException();
            }

            return searchResult;
        }
    }
}
```

Файл LicenseManagementService.cs

```
using Common.Entities;
using Common.Exceptions;
using Common.Mappers;
using LicenseManagementLambda.Interfaces;
using LicenseManagementLambda.Options;
using Microsoft.Extensions.Options;

namespace LicenseManagementLambda.Services
{
    /// <summary>
    /// Manages license CRUD requests.
    /// </summary>
    public class LicenseManagementService : ILicenseManagementService
    {
        private readonly ILicenseRepository _licenseRepository;
        private readonly HttpClient _httpClient;
        private ILogger<LicenseManagementService> _logger;

        /// <summary>
        /// Initializes a new instance of <see cref="LicenseManagementService"/>
class.
        /// </summary>
        /// <param name="licenseRepository"><see cref="ILicenseRepository"/></param>
        /// <param name="httpCLientFactory"><see cref="IHttpClientFactory"/></param>
        /// <param name="lambdaParameters"><see cref="LambdaParameters"/></param>
```

```
        /// <param name="logger">Logger instance.</param>
        public LicenseManagementService(ILicenseRepository licenseRepository,
IHttpClientFactory httpCLientFactory, IOptions<LambdaParameters> lambdaParameters,
ILogger<LicenseManagementService> logger)
        {
            _licenseRepository = licenseRepository;
            _httpClient = httpCLientFactory.CreateClient("ProductsAPI");
            _logger = logger;
        }

        /// <inheritdoc/>
        public async Task<LicenseDto> CreateLicenseAsync(Guid productId,
LicenseCreateModel licenseModel)
        {
            _logger.LogDebug("Trying to create a new license entity from: {@model}",
licenseModel);

            _logger.LogDebug("Checking the product. Target URL is {httpClientUrl}",
_httpClient.BaseAddress);
            var response = await _httpClient.GetAsync($"products?id={productId}");
            _logger.LogInformation("Received http response from products API:
{@response}", response.StatusCode);

            if (!response.IsSuccessStatusCode)
            {
                throw new ProductNotFoundException("Unable to create license:
product doesn't exist");
            }

            var licenseDto = licenseModel.MapToDto(productId);
            return await _licenseRepository.SaveAsync(licenseDto);
        }

        /// <inheritdoc/>
        public async Task<LicenseDto> DeleteLicenseAsync(Guid licenseId)
        {
            return await _licenseRepository.DeleteAsync(licenseId);
        }

        /// <inheritdoc/>
        public async Task<LicenseDto> GetLicenseByIdAsync(Guid licenseId)
        {
            var license = await _licenseRepository.GetByIdAsync(licenseId);

            if (license is null)
            {
                throw new LicenseNotFoundException();
            }

            return license;
        }

        /// <inheritdoc/>
        public async Task<LicenseDto> UpdateLicenseAsync(LicenseDto licenseDto)
        {
            var currentLicense = await
_licenseRepository.GetByIdAsync(licenseDto.LicenseId);

            if (currentLicense is null) throw new LicenseNotFoundException();

            if (licenseDto.ProductId != currentLicense.ProductId)
            {
                _logger.LogDebug("Product update requested. Checking the product
with ID: {id}", currentLicense.ProductId);
```

```
                var response = await
_httpClient.GetAsync($"products?id={currentLicense.ProductId}");
                _logger.LogDebug("Received http response from products API:
{@response}", response);

                if (!response.IsSuccessStatusCode)
                {
                    throw new ProductNotFoundException("Unable to update license:
product doesn't exist");
                }
            }

            return await _licenseRepository.SaveAsync(licenseDto);
        }
    }
}
```

Файл ProductEntitlementManagementService.cs

```
using Common.Constants;
using Common.Entities;
using Common.Exceptions;
using LicenseManagementLambda.Interfaces;
using Newtonsoft.Json;

namespace LicenseManagementLambda.Services
{
    public class ProductEntitlementManagementService :
IProductEntitlementManagementService
    {
        private readonly IProductEntitlementRepository
_productEntitlementRepository;
        private readonly ILicenseRepository _licenseRepository;
        private readonly HttpClient _productsHttpClient;
        private readonly HttpClient _usersHttpClient;
        private readonly ILogger<ProductEntitlementManagementService> _logger;

        /// <summary>
        /// Intiializes a new instance of <see
cref="ProductEntitlementManagementService"/> class.
        /// </summary>
        /// <param name="productEntitlementRepository"><see
cref="IProductEntitlementRepository"/></param>
        /// <param name="licenseRepository"><see cref="ILicenseRepository"/></param>
        /// <param name="httpClientFactory"><see cref="IHttpClientFactory"/></param>
        /// <param name="logger">Logger instance.</param>
        public ProductEntitlementManagementService(
            IProductEntitlementRepository productEntitlementRepository,
            ILicenseRepository licenseRepository,
            IHttpClientFactory httpClientFactory,
            ILogger<ProductEntitlementManagementService> logger)
        {
            _productEntitlementRepository = productEntitlementRepository;
            _licenseRepository = licenseRepository;
            _productsHttpClient = httpClientFactory.CreateClient("ProductsAPI");
            _usersHttpClient = httpClientFactory.CreateClient("UsersAPI");
            _logger = logger;
        }

        /// <inheritdoc/>
        public async Task<ProductEntitlementDto> CreateEntitlementAsync(Guid
licenseId, Guid userId)
```

```
        {
            var license = await _licenseRepository.GetByIdAsync(licenseId);
            if (license is null) throw new LicenseNotFoundException();

            var productsResponse = await
_productsHttpClient.GetAsync($"products?id={license.ProductId}");
            productsResponse.EnsureSuccessStatusCode();

            _logger.LogInformation("Received http response from Products API:
{response}", productsResponse.StatusCode);

            var productDetails = JsonConvert.DeserializeObject<ProductDto>(await
productsResponse.Content.ReadAsStringAsync());
            if (productDetails is null)
            {
                throw new ProductNotFoundException("Entitlement creation failed.
Unable to deserialize product entity.");
            }
            _logger.LogInformation("Received entity: {@response}", productDetails);

            var usersResponse = await
_usersHttpClient.GetAsync($"users?id={userId}");
            usersResponse.EnsureSuccessStatusCode();
            _logger.LogInformation("Received http response from Users API:
{response}", usersResponse.StatusCode);

            var userDetails = JsonConvert.DeserializeObject<UserDto>(await
usersResponse.Content.ReadAsStringAsync());
            if (userDetails is null)
            {
                throw new UserNotFoundException("Entitlement creation failed. Unable
to deserialize user entity.");
            }
            _logger.LogInformation("Received entity: {@response}", userDetails);

            ProductEntitlementDto productEntitlementDto = new()
            {
                UserId = userId.ToString(),
                LicenseId = licenseId.ToString(),
                ProductName = productDetails.Name,
                ProductDescription = productDetails.Description,
                UserName = userDetails.Username
            };
            _logger.LogInformation("Successfully created a new entitlement entity:
{@entitlement}", productEntitlementDto);

            return await
_productEntitlementRepository.SaveAsync(productEntitlementDto);
        }

        /// <inheritdoc/>
        public async Task<ProductEntitlementDto> DeleteEntitlementAsync(Guid
entitlementId)
        {
            return await _productEntitlementRepository.DeleteAsync(entitlementId);
        }

        /// <inheritdoc/>
        public async Task<ProductEntitlementDto> GetEntitlementByIdAsync(Guid
entitlementId)
        {
            return await _productEntitlementRepository.GetByIdAsync(entitlementId);
        }
```

```
        /// <inheritdoc/>
        public async Task<ProductEntitlementDto>
UpdateEntitlementAsync(ProductEntitlementDto entitlementDto)
        {
            return await _productEntitlementRepository.SaveAsync(entitlementDto);
        }

        public async Task UpdateUserDetails(BaseMessage<UserDto> details)
        {
            _logger.LogDebug("Received user details: {@details}", details);
            UserDto content = details.Content;
            IList<ProductEntitlementDto> entitlements = await
_productEntitlementRepository.GetByUserIdAsync(new Guid(details.EntityId));

            switch (details.Action)
            {
                case ProcessAction.Delete:
                    foreach (ProductEntitlementDto entry in entitlements)
                    {
                        entry.UserId = EntityTypes.Deleted;
                        entry.UserName = EntityTypes.Deleted;
                        await _productEntitlementRepository.SaveAsync(entry);
                    }
                    break;

                case ProcessAction.Update:
                    foreach (ProductEntitlementDto entry in entitlements)
                    {
                        entry.UserId = details.EntityId;
                        entry.UserName = content.Username;
                        await _productEntitlementRepository.SaveAsync(entry);
                    }
                    break;

                default:
                    break;
            }
        }

        public async Task UpdateProductDetails(BaseMessage<ProductDto> details)
        {
            _logger.LogDebug("Received product details: {@details}", details);
            ProductDto content = details.Content;
            IList<ProductEntitlementDto> entitlements = await
_productEntitlementRepository.GetByProductIdAsync(new Guid(details.EntityId));

            switch (details.Action)
            {
                case ProcessAction.Delete:
                    foreach (ProductEntitlementDto entry in entitlements)
                    {
                        entry.ProductDescription = EntityTypes.Deleted;
                        entry.ProductName = EntityTypes.Deleted;
                        await _productEntitlementRepository.SaveAsync(entry);
                    }
                    break;

                case ProcessAction.Update:
                    foreach (ProductEntitlementDto entry in entitlements)
                    {
                        entry.ProductDescription = content.Description;
                        entry.ProductName = content.Name;
                        await _productEntitlementRepository.SaveAsync(entry);
                    }
```

```
                    break;

                default:
                    break;
            }
        }
    }
}
```

Файл SqsEventProcessingService.cs

```
using Amazon.Lambda.SQSEvents;
using Common.Constants;
using Common.Entities;
using Common.Interfaces;
using LicenseManagementLambda.Interfaces;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using static Amazon.Lambda.SQSEvents.SQSEvent;

namespace LicenseManagementLambda.Services
{
    /// <summary>
    /// SQS Event processor.
    /// </summary>
    internal class SqsEventProcessingService : ISqsEventProcessingService
    {
        private readonly IProductEntitlementManagementService _entitlementService;
        private readonly ILogger<SqsEventProcessingService> _logger;

        /// <summary>
        /// Initializes a new instance of <see cref="SqsEventProcessingService"/>
class.
        /// </summary>
        /// <param name="strategySelector"><see
cref="IDataHandlerStrategySelector"/></param>
        /// <param name="logger">Logger instance.</param>
        public SqsEventProcessingService(IProductEntitlementManagementService
entitlementService, ILogger<SqsEventProcessingService> logger)
        {
            _entitlementService = entitlementService;
            _logger = logger;
        }

        /// <inheritdoc/>
        public Task ProcessAsync(JObject input)
        {
            if (input is null)
            {
                throw new ArgumentNullException(nameof(input));
            }

            return ProcessInternalAsync(input);
        }

        private async Task ProcessInternalAsync(JObject input)
        {
            _logger.LogDebug("SQS message processing started. Message: {message}",
input.ToString());
            var sqsEvent = input.ToObject<SQSEvent>();

            foreach (SQSMessage message in sqsEvent.Records)
            {
                _logger.LogDebug("Started processing SQSMessage {MessageId}",
message.MessageId);
```

```
                var body = JObject.Parse(message.Body);
                var entityType = body["EntityType"].ToString();
                _logger.LogDebug("Entity type is {type}", entityType);

                switch (entityType)
                {
                    case EntityTypes.User:
                        BaseMessage<UserDto> userMessage =
JsonConvert.DeserializeObject<BaseMessage<UserDto>>(message.Body);
                        _entitlementService.UpdateUserDetails(userMessage);
                        break;
                    case EntityTypes.Product:
                        BaseMessage<ProductDto> productMessage =
JsonConvert.DeserializeObject<BaseMessage<ProductDto>>(message.Body);
                        _entitlementService.UpdateProductDetails(productMessage);
                        break;
                }

                _logger.LogInformation("Processed SQSMessage {MessageId}",
message.MessageId);
            }
        }
    }
}
```

Файл Function.cs

```
using Amazon.Lambda.APIGatewayEvents;
using Amazon.Lambda.Core;
using Amazon.Lambda.Serialization.Json;
using Amazon.Lambda.SQSEvents;
using Common.Interfaces;
using LicenseManagementLambda.Builders;
using Newtonsoft.Json.Linq;

namespace LicenseManagementLambda
{
    public class Function
    {
        /// <summary>
        /// Default constructor. This constructor is used by Lambda to construct the
instance. When invoked in a Lambda environment
        /// the AWS credentials will come from the IAM role associated with the
function and the AWS region will be set to the
        /// region the Lambda function is executed in.
        /// </summary>
        public Function() { }

        /// <summary>
        /// This method is called for every Lambda invocation. This method takes in
an SQS event object and can be used
        /// to respond to SQS messages.
        /// </summary>
        /// <param name="input"></param>
        /// <param name="lambdaContext"></param>
        /// <returns></returns>
        [LambdaSerializer(typeof(JsonSerializer))]
        public async Task<APIGatewayProxyResponse> FunctionHandlerAsync(JObject
input, ILambdaContext lambdaContext)
        {
            var sqsEvent = input.ToObject<SQSEvent>();
            var request = input.ToObject<APIGatewayProxyRequest>();
```

```
            if (sqsEvent.Records is not null)
            {
                var services = new ServiceCollection();
                IServiceProvider _serviceProvider = new
ServiceProviderBuilder().Build(services);

                var service =
_serviceProvider.GetRequiredService<ISqsEventProcessingService>();

                await service.ProcessAsync(input);

                return new APIGatewayProxyResponse
                {
                    StatusCode = 200,
                };
            }
            if (request.Resource is not null)
            {
                LambdaEntryPoint lambdaEntryPoint = new();
                return await lambdaEntryPoint.FunctionHandlerAsync(request,
lambdaContext);
            }
            else
            {
                throw new ArgumentException("Input type is unknown and can't be
processed.");
            }
        }
    }
}
```

Файл Startup.cs

```
using Common.Extensions;
using Common.Middleware;
using LicenseManagementLambda.Interfaces;
using LicenseManagementLambda.Options;
using LicenseManagementLambda.Repositories;
using LicenseManagementLambda.Services;
using Microsoft.OpenApi.Models;

namespace LicenseManagementLambda;

/// <summary>
/// Startup class.
/// </summary>
public class Startup
{
    private readonly IConfiguration _configuration;

    /// <summary>
    /// Intializes a new instance of <see cref="Startup"/> class.
    /// </summary>
    /// <param name="configuration"><see cref="IConfiguration"/></param>
    /// <exception cref="ArgumentNullException"></exception>
    public Startup(IConfiguration configuration)
    {
        _configuration = configuration ?? throw new
ArgumentNullException(nameof(configuration));
    }

    /// <summary>
    /// Adds services to the DI container. This method gets called by the runtime.
```

```
    /// </summary>
    /// <param name="services"><see cref="IServiceCollection"/></param>
    public void ConfigureServices(IServiceCollection services)
    {
        services.ConfigureLambdaVariables<LambdaParameters>(_configuration);

        services.ConfigureLogging();
        services.ConfigureDynamoDB(_configuration);
        services.AddControllers();
        services.ConfigureSwaggerServices(new OpenApiInfo
        {
            Version = "v1",
            Title = "License Management Lambda",
            Description = "License API Lambda implementation for License Management
Service."
        });

        services.AddHttpClient("ProductsAPI", httpClient =>
        {
            httpClient.BaseAddress = new
Uri(_configuration.GetSection("Parameters:ProductsApiUrl").Value);
            httpClient.DefaultRequestHeaders.Add("Accept", "application/json");
        });
        services.AddHttpClient("UsersAPI", httpClient =>
        {
            httpClient.BaseAddress = new
Uri(_configuration.GetSection("Parameters:UsersApiUrl").Value);
            httpClient.DefaultRequestHeaders.Add("Accept", "application/json");
        });

        services.AddScoped<ILicenseManagementService, LicenseManagementService>();
        services.AddScoped<ILicenseRepository, LicenseRepository>();
        services.AddScoped<IProductEntitlementManagementService,
ProductEntitlementManagementService>();
        services.AddScoped<IProductEntitlementRepository,
ProductEntitlementRepository>();
    }

    /// <summary>
    /// Adds services to the DI container. This method gets called by the runtime.
    /// </summary>
    /// <param name="app"><see cref="IApplicationBuilder"/></param>
    public void Configure(IApplicationBuilder app)
    {
        app.UseHttpsRedirection();

        app.UseRouting();

        app.UseMiddleware<UnhandledExceptionLoggingMiddleware>();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllerRoute(
                name: "default",
                pattern: "license-api/{controller}/{id?}");
        });

        app.UseSwagger("license-api", _configuration);
    }
}
```

Файл ProductController.cs

```
using Common.Entities;
using Microsoft.AspNetCore.Mvc;
```

```
using ProductManagementLambda.Interfaces;
using Swashbuckle.AspNetCore.Annotations;
using System.ComponentModel.DataAnnotations;

namespace ProductManagementLambda.Controllers;

/// <summary>
/// API controller for products management in License Management Service.
/// </summary>
[ApiController]
[Route("products-api/products")]
[Produces("application/json")]
public class ProductController : ControllerBase
{
    private readonly IProductManagementService _productManagementService;
    private readonly ILogger<ProductController> _logger;

    /// <summary>
    /// Initializes a new instance of <see cref="ProductController"/> class.
    /// </summary>
    /// <param name="productManagementService"><see
cref="IProductManagementService"/></param>
    /// <param name="logger">Logger instance.</param>
    public ProductController(IProductManagementService productManagementService,
ILogger<ProductController> logger)
    {
        _productManagementService = productManagementService;
        _logger = logger;
    }

    /// <summary>
    /// Gets product by ID.
    /// </summary>
    /// <param name="id">Product's unique identifier.</param>
    /// <returns>Product entity.</returns>
    /// <remarks>
    /// Example url call:
    ///
    /// GET <code>license-management/products-api/products?id=ebff8ad4-24f9-4be7-
a15d-529f64ede7c6</code>
    /// </remarks>
    [HttpGet]
    [SwaggerResponse(StatusCodes.Status201Created, "Successfully returned item",
typeof(ProductDto))]
    [SwaggerResponse(StatusCodes.Status400BadRequest, "Incorrect input field
value")]
    [SwaggerResponse(StatusCodes.Status404NotFound, "Item does not present in the
system")]
    [SwaggerResponse(StatusCodes.Status500InternalServerError, "Unhandled exception
occured")]
    public async Task<IActionResult> GetProduct([Required, FromQuery] Guid id)
    {
        var product = await _productManagementService.GetProductByIdAsync(id);

        return Ok(product);
    }

    /// <summary>
    /// Creates new product.
    /// </summary>
    /// <param name="productDto"></param>
    /// <returns>Created product definition.</returns>
    /// <remarks>
    /// Example url call:
```

```
    ///
    /// POST <code>license-management/products-api/products</code>
    /// </remarks>
    [HttpPost]
    [SwaggerResponse(StatusCodes.Status201Created, "Returned successfully created
item", typeof(ProductDto))]
    [SwaggerResponse(StatusCodes.Status400BadRequest, "Incorrect input field
value")]
    [SwaggerResponse(StatusCodes.Status500InternalServerError, "Unhandled exception
occured")]
    public async Task<IActionResult> CreateProduct([Required, FromBody] ProductDto
productDto)
    {
        var product = await
_productManagementService.CreateProductAsync(productDto);

        return CreatedAtAction(nameof(_productManagementService.CreateProductAsync),
product);
    }

    /// <summary>
    /// Updates an exisiting product.
    /// </summary>
    /// <param name="productDto"><see cref="ProductDto"/></param>
    /// <returns>Updated product definition.</returns>
    /// <remarks>
    /// Example url call:
    ///
    /// PUT <code>license-management/products-api/products</code>
    /// </remarks>
    [HttpPut]
    [SwaggerResponse(StatusCodes.Status201Created, "Returned successfully updated
item", typeof(ProductDto))]
    [SwaggerResponse(StatusCodes.Status400BadRequest, "Incorrect input field
value")]
    [SwaggerResponse(StatusCodes.Status404NotFound, "Item does not present in the
system")]
    [SwaggerResponse(StatusCodes.Status500InternalServerError, "Unhandled exception
occured")]
    public async Task<IActionResult> UpdateProduct([Required, FromBody] ProductDto
productDto)
    {
        var product = await
_productManagementService.UpdateProductAsync(productDto);

        return CreatedAtAction(nameof(_productManagementService.UpdateProductAsync),
product);
    }

    /// <summary>
    /// Deletes an existing product.
    /// </summary>
    /// <param name="id">Product unique identifier.</param>
    /// <returns>Deleted product definition.</returns>
    /// <remarks>
    /// Example url call:
    ///
    /// DELETE <code>license-management/products-api/products?id=ebff8ad4-24f9-4be7-
a15d-529f64ede7c6</code>
    /// </remarks>
    [HttpDelete]
    [SwaggerResponse(StatusCodes.Status200OK, "Returned successfully deleted item",
typeof(ProductDto))]
```

```
    [SwaggerResponse(StatusCodes.Status400BadRequest, "Incorrect input field
value")]
    [SwaggerResponse(StatusCodes.Status404NotFound, "Item does not present in the
system")]
    [SwaggerResponse(StatusCodes.Status500InternalServerError, "Unhandled exception
occured")]
    public async Task<IActionResult> DeleteProduct([Required, FromQuery] Guid id)
    {
        var product = await _productManagementService.DeleteProductAsync(id);

        return Ok(product);
    }
}
```

Файл IProductManagementService.cs

```
using Common.Entities;

namespace ProductManagementLambda.Interfaces
{
    /// <summary>
    /// Interface of product CRUD management service.
    /// </summary>
    public interface IProductManagementService
    {
        /// <summary>
        /// Gets product from the datastore.
        /// </summary>
        /// <param name="productId">Product's unqiue indentifier.</param>
        /// <returns>Entity.</returns>
        Task<ProductDto> GetProductByIdAsync(Guid productId);

        /// <summary>
        /// Manages product creation operation.
        /// </summary>
        /// <param name="productDto"><see cref="ProductDto"/></param>
        /// <returns>Created entity.</returns>
        Task<ProductDto> CreateProductAsync(ProductDto productDto);

        /// <summary>
        /// Manages product deletion operation.
        /// </summary>
        /// <param name="productId"><see cref="ProductDto"/></param>
        /// <returns>Deleted entity.</returns>
        Task<ProductDto> DeleteProductAsync(Guid productId);

        /// <summary>
        /// Manages product update operation.
        /// </summary>
        /// <param name="productDto"><see cref="ProductDto"/></param>
        /// <returns>Updated entity.</returns>
        Task<ProductDto> UpdateProductAsync(ProductDto productDto);
    }
}
```

Файл ProductManagementService.cs

```
using Common.Entities;
using Common.Exceptions;
using ProductManagementLambda.Interfaces;

namespace ProductManagementLambda.Repositories
{
```

```
    /// <summary>
    /// Manages product operations.
    /// </summary>
    public class ProductManagementService : IProductManagementService
    {
        private readonly IProductRepository _productRepository;
        private readonly ILogger<ProductManagementService> _logger;

        /// <summary>
        /// Initializes a new instance of <see cref="ProductManagementService"/>
class.
        /// </summary>
        /// <param name="productRepository"><see cref="IProductRepository"/></param>
        /// <param name="logger">Logger instance.</param>
        public ProductManagementService(IProductRepository productRepository,
ILogger<ProductManagementService> logger)
        {
            _productRepository = productRepository;
            _logger = logger;
        }

        /// <inheritdoc/>
        public async Task<ProductDto> CreateProductAsync(ProductDto productDto)
        {
            return await _productRepository.SaveAsync(productDto);
        }

        /// <inheritdoc/>
        public async Task<ProductDto> DeleteProductAsync(Guid productId)
        {
            return await _productRepository.DeleteAsync(productId);
        }

        /// <inheritdoc/>
        public async Task<ProductDto> GetProductByIdAsync(Guid productId)
        {
            var product = await _productRepository.GetByIdAsync(productId);

            if (product is null)
            {
                throw new ProductNotFoundException();
            }

            return product;
        }

        /// <inheritdoc/>
        public async Task<ProductDto> UpdateProductAsync(ProductDto productDto)
        {
            var product = await
_productRepository.GetByIdAsync(productDto.ProductId);

            if (product is null) throw new ProductNotFoundException();

            return await _productRepository.SaveAsync(productDto);
        }
    }
}
```

Файл ProductRepository.cs

```
using Amazon.DynamoDBv2.DataModel;
using Common.Entities;
```

```
using Common.Exceptions;
using ProductManagementLambda.Interfaces;

namespace ProductManagementLambda.Repositories
{
    /// <summary>
    /// Products datastore service.
    /// </summary>
    public class ProductRepository : IProductRepository
    {
        private readonly IDynamoDBContext _dynamoDbContext;
        private readonly ILogger<ProductRepository> _logger;

        /// <summary>
        /// Initializes a new instance of <see cref="ProductRepository"/> class.
        /// </summary>
        /// <param name="dynamoDbContext"><see cref="IDynamoDBContext"/></param>
        /// <param name="logger">Logger instance.</param>
        public ProductRepository(IDynamoDBContext dynamoDbContext,
ILogger<ProductRepository> logger)
        {
            _dynamoDbContext = dynamoDbContext;
            _logger = logger;
        }

        /// <inheritdoc/>
        public async Task<ProductDto> GetByIdAsync(Guid id)
        {
            _logger.LogDebug("Trying to get product entity with id: {id}", id);

            ProductDto product = await _dynamoDbContext.LoadAsync<ProductDto>(id);

            _logger.LogInformation("Successfully retrieved product entity:
{@entity}", product);

            return product;
        }

        /// <inheritdoc/>
        public async Task<ProductDto> DeleteAsync(Guid id)
        {
            _logger.LogDebug("Trying to delete product entity with id: {id}", id);

            var config = new DynamoDBOperationConfig
            {
                IgnoreNullValues = false
            };

            ProductDto product = await _dynamoDbContext.LoadAsync<ProductDto>(id,
config);
            if (product == null)
            {
                throw new ProductNotFoundException();
            }

            await _dynamoDbContext.DeleteAsync<ProductDto>(id);
            _logger.LogInformation("Successfully deleted product entity:
{@entity}}", product);

            return product;
        }

        /// <inheritdoc/>
        public async Task<ProductDto> SaveAsync(ProductDto product)
```

```
        {
            _logger.LogDebug("Trying to save product entity: {@entity}", product);

            var config = new DynamoDBOperationConfig
            {
                IgnoreNullValues = false
            };

            await _dynamoDbContext.SaveAsync(product, config);
            _logger.LogInformation("Successfully save a product entity: {@entity}",
product);

            return product;
        }
    }
}
```

Файл Startup.cs

```
using Common.Extensions;
using Microsoft.OpenApi.Models;
using ProductManagementLambda.Interfaces;
using ProductManagementLambda.Repositories;
using System.Diagnostics.CodeAnalysis;

namespace ProductManagementLambda;

/// <summary>
/// Startup class.
/// </summary>
[ExcludeFromCodeCoverage]
public class Startup
{
    private IConfiguration _configuration;

    /// <summary>
    /// Intializes a new instance of <see cref="Startup"/> class.
    /// </summary>
    /// <param name="configuration"><see cref="IConfiguration"/></param>
    /// <exception cref="ArgumentNullException"></exception>
    public Startup(IConfiguration configuration)
    {
        _configuration = configuration ?? throw new
ArgumentNullException(nameof(configuration));
    }

    /// <summary>
    /// Adds services to the DI container. This method gets called by the runtime.
    /// </summary>
    /// <param name="services"><see cref="IServiceCollection"/></param>
    public void ConfigureServices(IServiceCollection services)
    {
        services.ConfigureLogging();
        services.ConfigureDynamoDB(_configuration);
        services.AddControllers();
        services.ConfigureSwaggerServices(new OpenApiInfo
        {
            Version = "v1",
            Title = "Product Management Lambda",
            Description = "Products API Lambda implementation for License Management
Service."
        });
```

```
            services.AddScoped<IProductManagementService, ProductManagementService>();
            services.AddScoped<IProductRepository, ProductRepository>();
        }

        /// <summary>
        /// Adds services to the DI container. This method gets called by the runtime.
        /// </summary>
        /// <param name="app"><see cref="IApplicationBuilder"/></param>
        public void Configure(IApplicationBuilder app)
        {
            app.UseHttpsRedirection();

            app.UseRouting();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "products-api/{controller}/{id?}");
            });

            app.UseSwagger("products-api", _configuration);
        }
    }
```

Файл ServiceProviderBuilder.cs

```
using Common.Extensions;
using Common.Interfaces;
using Common.Utils;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using System.Diagnostics.CodeAnalysis;
using UserIntegrationLambda.Extensions;
using UserIntegrationLambda.InputProcessStrategies;
using UserIntegrationLambda.Interfaces;
using UserIntegrationLambda.Options;
using UserIntegrationLambda.Repository;
using UserIntegrationLambda.Services;

namespace UserIntegrationLambda.Builders
{
    /// <summary>
    /// Provides a creation service of <see cref="IServiceProvider"/> instance.
    /// </summary>
    [ExcludeFromCodeCoverage]
    public class ServiceProviderBuilder
    {
        private readonly IConfiguration _configuration;

        public ServiceProviderBuilder()
        {
            _configuration = new ConfigurationBuilder()
                .SetBasePath(Directory.GetCurrentDirectory())
                .AddJsonFile("appsettings.json", optional: false, reloadOnChange:
true)
                .AddEnvironmentVariables()
                .Build();
        }

        /// <summary>
        /// Creates a <see cref="IServiceProvider"/> containing services provided
for solution.
        /// </summary>
        /// <param name="serviceCollection"></param>
```

```
        /// <returns><see cref="ServiceProvider"/></returns>
        /// <exception cref="ArgumentNullException"></exception>
        public IServiceProvider Build(IServiceCollection serviceCollection)
        {
            if (serviceCollection == null)
            {
                throw new ArgumentNullException(nameof(serviceCollection));
            }

            serviceCollection.ConfigureLogging();


serviceCollection.ConfigureLambdaVariables<LambdaParameters>(_configuration);
            serviceCollection.ConfigureDynamoDB(_configuration);
            serviceCollection.ConfigureCircuitBreakerServices(_configuration);

            AddServices(serviceCollection);

            var serviceProvider = serviceCollection.BuildServiceProvider();

            return serviceProvider;
        }

        private static void AddServices(IServiceCollection services)
        {
            var lambdaContextAccessor = new LambdaContextAccessor();
            services.AddSingleton<ILambdaContextAccessor>(lambdaContextAccessor);
            services.AddSingleton(lambdaContextAccessor);

            services.AddScoped<ISqsEventProcessingService,
SqsEventProcessingService>();
            services.AddScoped<IDataHandlerStrategySelector,
DataHandlerStrategySelector>();
            services.AddScoped<ISqsRecordProcessingService,
SqsRecordProcessingService>();
            services.AddScoped<IUserIntegrationHandler, UserIntegrationHandler>();

            services
                .AddScoped<IDataHandlerStrategy, UserIntegrationHandlerStrategy>()
                .AddScoped<IDataHandlerStrategy,
CircuitBreakerMessageHandlerStrategy>();

            services.AddScoped<IUsersWriteRepository, UsersWriteRepository>();
        }
    }
}
```

Файл NoSuitableHandlerException.cs

```
using System.Runtime.Serialization;

namespace UserIntegrationLambda.Exceptions
{
    /// <summary>
    /// Represents absence of suitable handler for SQS message.
    /// </summary>
    [Serializable]
    internal class NoSuitableHandlerException : Exception
    {
        /// <summary>
        /// Initializes a new instance of <see cref="NoSuitableHandlerException"/>
class.
        /// </summary>
```

```
        public NoSuitableHandlerException() : base("No appropriate handler has been
found for the incoming message type.")
        {
        }

        /// <summary>
        /// Initializes a new instance of <see cref="NoSuitableHandlerException"/>
class.
        /// </summary>
        /// <param name="message">Exception message.</param>
        public NoSuitableHandlerException(string? message) : base(message)
        {
        }

        /// <summary>
        /// Initializes a new instance of <see cref="NoSuitableHandlerException"/>
class.
        /// </summary>
        /// <param name="message">Exception message.</param>
        /// <param name="innerException">Inner exception.</param>
        public NoSuitableHandlerException(string? message, Exception?
innerException) : base(message, innerException)
        {
        }

        /// <summary>
        /// Initializes a new instance of <see cref="NoSuitableHandlerException"/>
class.
        /// </summary>
        /// <param name="info"><see cref="SerializationInfo"/></param>
        /// <param name="context"><see cref="StreamingContext"/></param>
        protected NoSuitableHandlerException(SerializationInfo info,
StreamingContext context) : base(info, context)
        {
        }
    }
}
```

Файл CircuitBreakerConfigurationExtensions.cs

```
using Amazon.EventBridge;
using Amazon.Lambda;
using Amazon.SQS;
using Common.Interfaces;
using Common.Options;
using Common.Services;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using UserIntegrationLambda.Interfaces.CircuitBreaker;
using UserIntegrationLambda.Options;
using UserIntegrationLambda.Repository;
using UserIntegrationLambda.Services.CircuitBreaker;

namespace UserIntegrationLambda.Extensions
{
    public static class CircuitBreakerConfigurationExtensions
    {
        public static IServiceCollection ConfigureCircuitBreakerServices(this
IServiceCollection serviceCollection, IConfiguration configuration)
        {

serviceCollection.Configure<CircuitBreakerOptions>(configuration.GetSection("Circuit
Breaker"));
```

```
serviceCollection.Configure<EventBridgeOptions>(configuration.GetSection("EventBridg
e"));

            serviceCollection.AddAWSService<IAmazonEventBridge>();
            serviceCollection.AddAWSService<IAmazonSQS>();

            serviceCollection.AddScoped<ICircuitStateRepository,
CircuitStateRepository>();
            serviceCollection.AddScoped<ICircuitStateToDatabaseDtoMapper,
CircuitStateToDatabaseDtoMapper>();

            serviceCollection
                .AddScoped<IEventBridgeSchedulerService,
EventBridgeSchedulerService>()
                .AddScoped<IAmazonLambda, AmazonLambdaClient>()
                .AddScoped<ISqsClient, SqsClient>()
                .AddScoped<IEventSourceMappingClient,
SqsEventSourceMappingClient>();

            serviceCollection.AddScoped<ICircuitBreakingService,
CircuitBreakingService>();

            return serviceCollection;
        }
    }
}
```

Файл CircuitBreakerMessageHandlerStrategy.cs

```
using Microsoft.Extensions.Logging;
using Newtonsoft.Json.Linq;
using UserIntegrationLambda.Interfaces;
using UserIntegrationLambda.Interfaces.CircuitBreaker;
using UserIntegrationLambda.Models;

namespace UserIntegrationLambda.InputProcessStrategies
{
    /// <summary>
    /// Processes circuit breaker maintenance messages.
    /// </summary>
    public class CircuitBreakerMessageHandlerStrategy : IDataHandlerStrategy
    {
        private readonly ICircuitBreakingService _circuitBreakingService;
        private readonly ISqsRecordProcessingService _sqsRecordProcessingService;
        private readonly ILogger<CircuitBreakerMessageHandlerStrategy> _logger;

        /// <summary>
        /// Initializes a new instance of <see
cref="CircuitBreakerMessageHandlerStrategy"/> class.
        /// </summary>
        /// <param name="circuitBreakingService"><see
cref="ICircuitBreakingService"/></param>
        /// <param name="sqsRecordProcessingService"><see
cref="ISqsRecordProcessingService"/></param>
        /// <param name="logger">Logger instance.</param>
        public CircuitBreakerMessageHandlerStrategy(ICircuitBreakingService
circuitBreakingService, ISqsRecordProcessingService sqsRecordProcessingService,
ILogger<CircuitBreakerMessageHandlerStrategy> logger)
        {
            _circuitBreakingService = circuitBreakingService;
            _sqsRecordProcessingService = sqsRecordProcessingService;
            _logger = logger;
```

```
        }

        /// <inheritdoc/>
        public bool IsSuitable(JObject input)
        {
            return input.ContainsKey("IsMaintenance") &&
input.Value<bool>("IsMaintenance");
        }

        /// <inheritdoc/>
        public async Task ProcessInputAsync(JObject input)
        {
            _logger.LogDebug("Received maintenance message.");

            if (input == null) throw new ArgumentNullException(nameof(input));

            CircuitBreakerMessage? message =
input.ToObject<CircuitBreakerMessage>();
            _logger.LogDebug("Deserialized circuit breaker message into:
{@message}", message);

            if (message == null) throw new ArgumentNullException(nameof(input));

            switch (message.Action)
            {
                case CircuitBreakerActions.Open:
                    await _circuitBreakingService.Open(message.Timeout);
                    break;
                case CircuitBreakerActions.Close:
                    await _circuitBreakingService.Close();
                    break;
                case CircuitBreakerActions.CircuitBreakerTrial:
                    await
_circuitBreakingService.Trial(_sqsRecordProcessingService.ProcessMessageAsync);
                    break;
                case CircuitBreakerActions.PermanentlyClose:
                    await _circuitBreakingService.PermanentlyClose();
                    break;
                default:
                    throw new NotSupportedException($"Action {message.Action} is not
supported.");
                }
        }
    }
}
```

Файл DataHandlerStrategySelector.cs

```
using Microsoft.Extensions.Logging;
using Newtonsoft.Json.Linq;
using UserIntegrationLambda.Exceptions;
using UserIntegrationLambda.Interfaces;

namespace UserIntegrationLambda.InputProcessStrategies
{
    /// <summary>
    /// Picks the correct strategy to handle specific input.
    /// </summary>
    public class DataHandlerStrategySelector : IDataHandlerStrategySelector
    {
        private readonly ILogger<DataHandlerStrategySelector> _logger;
        private readonly IEnumerable<IDataHandlerStrategy> _handlers;
```

```
        /// <summary>
        /// Initializes a new instance of <see cref="DataHandlerStrategySelector"/>
class.
        /// <param name="logger">Logger instance.</param>
        /// <param name="handlers">The collection of all available input
handlers.</param>
        /// </summary>
        public DataHandlerStrategySelector(ILogger<DataHandlerStrategySelector>
logger, IEnumerable<IDataHandlerStrategy> handlers)
        {
            _logger = logger;
            _handlers = handlers;
        }

        /// <inheritdoc/>
        public IDataHandlerStrategy GetStrategy(JObject input)
        {
            _logger.LogDebug("Getting strategy for input: {input}",
input.ToString());

            var strategy = _handlers.FirstOrDefault(h => h.IsSuitable(input));

            if (strategy is null)
            {
                throw new NoSuitableHandlerException();
            }

            _logger.LogInformation("Chosen input handler is {strategyHandler}",
strategy.GetType());
            return strategy;
        }
    }
}
```

Файл UserIntegrationHandlerStrategy.cs

```
using Amazon.Lambda.SQSEvents;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json.Linq;
using UserIntegrationLambda.Interfaces;

namespace UserIntegrationLambda.InputProcessStrategies
{
    /// <summary>
    /// Processes user integration requests SQS messages.
    /// </summary>
    public class UserIntegrationHandlerStrategy : IDataHandlerStrategy
    {
        private readonly ISqsRecordProcessingService _sqsRecordProcessingService;
        private readonly ILogger<UserIntegrationHandlerStrategy> _logger;

        /// <summary>
        /// Initializes a new instance of <see
cref="UserIntegrationHandlerStrategy"/> class.
        /// </summary>
        /// <param name="sqsRecordProcessingService"></param>
        /// <param name="logger"></param>
        public UserIntegrationHandlerStrategy(ISqsRecordProcessingService
sqsRecordProcessingService, ILogger<UserIntegrationHandlerStrategy> logger)
        {
            _sqsRecordProcessingService = sqsRecordProcessingService;
            _logger = logger;
        }
```

```
        /// <inheritdoc/>
        public bool IsSuitable(JObject input)
        {
            return input.ContainsKey("Records");
        }

        /// <inheritdoc/>
        public async Task ProcessInputAsync(JObject input)
        {
            var sqsEvent = input.ToObject<SQSEvent>();

            if (sqsEvent is null) throw new ArgumentNullException(nameof(input));
            _logger.LogDebug("Deserialized input into SQS Event: {@sqsEvent}",
sqsEvent);

            await _sqsRecordProcessingService.ProcessSqsRecordsAsync(sqsEvent);
        }
    }
}
```

Файл ICircuitBreakingService.cs

```
using static Amazon.Lambda.SQSEvents.SQSEvent;

namespace UserIntegrationLambda.Interfaces.CircuitBreaker
{
    /// <summary>
    /// Interface of a circuit breaker service adapter.
    /// </summary>
    public interface ICircuitBreakingService
    {
        /// <summary>
        /// Processes sqs message and breaks execution circuit if necessary.
        /// </summary>
        /// <param name="sqsMessage">Incoming sqs message.</param>
        /// <param name="action">Call to the external service.</param>
        /// <returns>Task.</returns>
        Task ExecuteAsync(SQSMessage sqsMessage, Func<SQSMessage, Task> action);

        /// <summary>
        /// Changes the circuit breaker to open state.
        /// </summary>
        /// <param name="customTimeout">Overrided value for circuit breaker
timeout.</param>
        /// <returns>Task.</returns>
        Task Open(TimeSpan? customTimeout);

        /// <summary>
        /// Changes the circuit breaker to closed state.
        /// </summary>
        /// <returns>Task.</returns>
        Task Close();

        /// <summary>
        /// Permanently close the circuit breaker.
        /// </summary>
        /// <returns>Task.</returns>
        Task PermanentlyClose();

        /// <summary>
        /// Changes the circuit breaker to half-open state.
        /// </summary>
        /// <returns>Task.</returns>
        Task HalfOpen();
```

```
        /// <summary>
        /// Tries execution of one message.
        /// </summary>
        /// <param name="action">Call to the external service.</param>
        /// <returns>Task.</returns>
        Task Trial(Func<SQSMessage, Task> action);
    }
}
```

Файл ICircuitStateRepository.cs

```
using Resiliency.CircuitBreaker;

namespace UserIntegrationLambda.Interfaces.CircuitBreaker
{
    /// <summary>
    /// Interface of Circuit Breaker State datastore.
    /// </summary>
    public interface ICircuitStateRepository
    {
        /// <summary>
        /// Gets the state of circuit breaker.
        /// </summary>
        /// <returns>Task.</returns>
        Task<CircuitState> GetAsync();

        /// <summary>
        /// Saves an item to the datastore.
        /// </summary>
        /// <param name="circuitState">New state of the circuit breaker.</param>
        /// <returns></returns>
        Task SaveAsync(CircuitState circuitState);
    }
}
```

Файл ICircuitStateToDatabaseDtoMapper.cs

```
using Resiliency.CircuitBreaker;
using UserIntegrationLambda.Models;

namespace UserIntegrationLambda.Interfaces.CircuitBreaker
{
    /// <summary>
    /// Interface for circuit state database DTO mapper.
    /// </summary>
    public interface ICircuitStateToDatabaseDtoMapper : ICircuitStateVisitor
    {
        /// <summary>
        /// Returns circuit state DTO.
        /// </summary>
        CircuitStateDatabaseDto? CircuitStateDatabaseDto { get; }
    }
}
```

Файл IDataHandlerStrategy.cs

```
using Newtonsoft.Json.Linq;

namespace UserIntegrationLambda.Interfaces
{
    /// <summary>
    /// Interface of a message handler for an appropriate incoming message type.
```

```
    /// </summary>
    public interface IDataHandlerStrategy
    {
        /// <summary>
        /// Runs input message handler.
        /// </summary>
        /// <param name="input">Incomming message.</param>
        /// <returns>Task.</returns>
        Task ProcessInputAsync(JObject input);

        /// <summary>
        /// Checks if this particular handler fits the input.
        /// </summary>
        /// <param name="input">Incomming JSON message.</param>
        /// <returns>True/False.</returns>
        bool IsSuitable(JObject input);
    }
}
```

Файл IDataHandlerStrategySelector.cs

```
using Newtonsoft.Json.Linq;

namespace UserIntegrationLambda.Interfaces
{
    /// <summary>
    /// Interface for strategy pattern selector service.
    /// </summary>
    public interface IDataHandlerStrategySelector
    {
        /// <summary>
        /// Picks strategy to process given input.
        /// </summary>
        /// <param name="input">JSON object.</param>
        /// <returns>Strategy to handle input JSON.</returns>
        IDataHandlerStrategy GetStrategy(JObject input);
    }
}
```

Файл ISqsRecordProcessingService.cs

```
using Amazon.Lambda.SQSEvents;
using static Amazon.Lambda.SQSEvents.SQSEvent;

namespace UserIntegrationLambda.Interfaces
{
    /// <summary>
    ///  Interface of SQS record processing service.
    /// </summary>
    public interface ISqsRecordProcessingService
    {
        /// <summary>
        /// Processes all records from an SQS Event.
        /// </summary>
        /// <param name="sQSEvent"><see cref="SQSEvent"/></param>
        /// <returns>Task.</returns>
        Task ProcessSqsRecordsAsync(SQSEvent sqsEvent);

        /// <summary>
        /// Processes one SQS message.
        /// </summary>
        /// <param name="sqsMessage"><see cref="SQSMessage"/></param>
        /// <returns>Task.</returns>
        Task ProcessMessageAsync(SQSMessage sqsMessage);
    }
```

```
}
```

Файл IUserIntegrationHandler.cs

```
using Common.Entities;

namespace UserIntegrationLambda.Interfaces
{
    /// <summary>
    /// Interface of user synchronization service between lambda and downstream
consumer.
    /// </summary>
    public interface IUserIntegrationHandler
    {
        /// <summary>
        /// Creates a new user.
        /// </summary>
        /// <param name="user">User to be created.</param>
        /// <returns></returns>
        Task CreateUser(UserDto user);

        /// <summary>
        /// Updates existing user.
        /// </summary>
        /// <param name="user">User to be updated.</param>
        /// <returns></returns>
        Task UpdateUser(UserDto user);

        /// <summary>
        /// Deletes existing user.
        /// </summary>
        /// <param name="uuid">User's ID.</param>
        /// <returns></returns>
        Task DeleteUser(Guid uuid);
    }
}
```

Файл CircuitBreakerActions.cs

```
namespace UserIntegrationLambda.Models
{
    /// <summary>
    /// Represents Circuit Breaker possible change actions.
    /// </summary>
    internal enum CircuitBreakerActions
    {
        Open,
        Close,
        CircuitBreakerTrial,
        PermanentlyClose,
        ReturnState,
        Retry
    }
}
```

Файл CircuitBreakerMessage.cs

```
namespace UserIntegrationLambda.Models
{
    /// <summary>
    /// Model used to describe Circuit Breaker maintenance message.
    /// </summary>
    internal class CircuitBreakerMessage
```

```
    {
        /// <summary>
        /// Gets or sets the flag if this the message is maintenance.
        /// </summary>
        public bool IsMaintenance { get; set; }

        /// <summary>
        /// Gets or sets the message action.
        /// </summary>
        public CircuitBreakerActions Action { get; set; }

        /// <summary>
        /// Gets or sets the Circuit Breaker timeout in minutes.
        /// </summary>
        public TimeSpan? Timeout { get; set; }
    }
}
```

Файл CircuitStateDatabaseDto.cs

```
using Amazon.DynamoDBv2.DataModel;

namespace UserIntegrationLambda.Models
{
    /// <summary>
    /// Model used to save the CircuitState into the database.
    /// </summary>
    [DynamoDBTable("LicenseManagement-CircuitBreakerState")]
    public class CircuitStateDatabaseDto
    {
        /// <summary>
        /// Entity ID value.
        /// </summary>
        [DynamoDBHashKey]
        public string Id { get; set; } = null!;

        /// <summary>
        /// Circuit Breaker state.
        /// </summary>
        public string StateId { get; set; } = null!;

        /// <summary>
        /// Opening date.
        /// </summary>
        public DateTime? OpenedAt { get; set; }

        /// <summary>
        /// Starting date.
        /// </summary>
        public DateTime? StartedAt { get; set; }

        /// <summary>
        /// Number of errors occured.
        /// </summary>
        public int? ErrorCounter { get; set; }
    }
}
```

Файл CircuitBreakerOptions.cs

```
namespace UserIntegrationLambda.Options
```

```
{
    /// <summary>
    /// Contains configuration parameters for circuit breaker.
    /// </summary>
    public class CircuitBreakerOptions
    {
        /// <summary>
        /// Name of circuit state Dynamo DB table.
        /// </summary>
        public string CircuitStateTableName { get; set; } = null!;

        /// <summary>
        /// Max number of errors before opening.
        /// </summary>
        public int ErrorThreshold { get; set; } = default!;

        /// <summary>
        /// Timeout value.
        /// </summary>
        public TimeSpan Timeout { get; set; } = default!;

        /// <summary>
        /// Sampling duration in minutes.
        /// </summary>
        public TimeSpan SamplingDuration { get; set; } = default!;

        /// <summary>
        /// Dead Letter Queue url.
        /// </summary>
        public Uri DeadLetterQueueUrl { get; set; } = null!;

        /// <summary>
        /// SQS source queue url.
        /// </summary>
        public Uri SourceQueueUrl { get; set; } = null!;
    }
}
```

Файл LambdaParameters.cs

```
namespace UserIntegrationLambda.Options
{
    /// <summary>
    /// Contains lambda parameters.
    /// </summary>
    public class LambdaParameters
    {
        /// <summary>
        /// Name of users Dynamo DB table.
        /// </summary>
        public string UsersTableName { get; set; } = null!;
    }
}
```

Файл CircuitStateRepository.cs

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using Resiliency.CircuitBreaker;
using UserIntegrationLambda.Interfaces.CircuitBreaker;
using UserIntegrationLambda.Models;
```

```
using UserIntegrationLambda.Options;

namespace UserIntegrationLambda.Repository
{
    /// <summary>
    /// Circuit breaker state datastore.
    /// </summary>
    public class CircuitStateRepository : ICircuitStateRepository
    {
        private static readonly string StateId = "7b5523ce-4b60-441d-98d5-
a23bde47f7ae";

        private readonly ICircuitStateToDatabaseDtoMapper _dbMapper;
        private readonly IDynamoDBContext _dynamoDBContext;
        private readonly CircuitBreakerOptions _options;
        private readonly ILogger<CircuitStateRepository> _logger;

        /// <summary>
        /// Initializes a new instance of <see cref="CircuitStateRepository"/>
class.
        /// </summary>
        /// <param name="dbMapper"><see
cref="ICircuitStateToDatabaseDtoMapper"/></param>
        /// <param name="dynamoDBContext"><see cref="IDynamoDBContext"/></param>
        /// <param name="circuitBreakerOptions"><see
cref="CircuitBreakerOptions"/></param>
        /// <param name="logger">Logger instance.</param>
        public CircuitStateRepository(ICircuitStateToDatabaseDtoMapper dbMapper,
IDynamoDBContext dynamoDBContext, IOptions<CircuitBreakerOptions>
circuitBreakerOptions, ILogger<CircuitStateRepository> logger)
        {
            _dbMapper = dbMapper;
            _dynamoDBContext = dynamoDBContext;
            _options = circuitBreakerOptions.Value;
            _logger = logger;
        }

        /// <inheritdoc/>
        public async Task<CircuitState> GetAsync()
        {
            var openStateConfig = new OpenStateConfig(_options.Timeout);
            var closedStateConfig = new ClosedStateConfig(_options.ErrorThreshold,
_options.SamplingDuration);

            try
            {
                CircuitStateDatabaseDto databaseDto = await
_dynamoDBContext.LoadAsync<CircuitStateDatabaseDto>(StateId) ?? GetClosedStateDto();

                CircuitState circuitState = databaseDto.StateId switch
                {
                    "Open" => new
OpenState(databaseDto.OpenedAt.GetValueOrDefault(DateTime.UtcNow), openStateConfig,
closedStateConfig),
                    "HalfOpen" => new HalfOpenState(openStateConfig,
closedStateConfig),
                    "Closed" => GetClosedState(databaseDto, openStateConfig,
closedStateConfig),
                    "PermanentlyClosed" => new PermanentlyClosedState(),
                    _ => throw new NotSupportedException($"{databaseDto.StateId} is
not supported.")
                };

                return circuitState;
```

```
            }
            catch (AmazonDynamoDBException ex)
            {
                _logger.LogCritical(ex, "Failed to load circuit state from database.
Falling back to default (Closed) state.");
                CircuitStateDatabaseDto closedStateDto = GetClosedStateDto();
                return GetClosedState(closedStateDto, openStateConfig,
closedStateConfig);
            }
        }

        /// <inheritdoc/>
        public async Task SaveAsync(CircuitState circuitState)
        {
            circuitState.Accept(_dbMapper);

            try
            {
                if (_dbMapper.CircuitStateDatabaseDto != null)
                {
                    _dbMapper.CircuitStateDatabaseDto.Id = StateId;
                    await
_dynamoDBContext.SaveAsync(_dbMapper.CircuitStateDatabaseDto);
                }
            }
            catch (AmazonDynamoDBException ex)
            {
                _logger.LogCritical(ex, "Failed to save circuit state to
database.");
            }
        }

        private static CircuitStateDatabaseDto GetClosedStateDto()
        {
            return new CircuitStateDatabaseDto
            {
                Id = StateId,
                StateId = "Closed"
            };
        }

        private static ClosedState GetClosedState(CircuitStateDatabaseDto
databaseDto, OpenStateConfig openStateConfig, ClosedStateConfig closedStateConfig)
        {
            return new ClosedState(databaseDto.ErrorCounter.GetValueOrDefault(0),
databaseDto.StartedAt.GetValueOrDefault(DateTime.UtcNow), openStateConfig,
closedStateConfig);
        }
    }
}
```

Файл UsersWriteRepository.cs

```
using Amazon.DynamoDBv2.DataModel;
using Common.Entities;
using Common.Exceptions;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using UserIntegrationLambda.Interfaces;
using UserIntegrationLambda.Options;

namespace UserIntegrationLambda.Repository
{
```

```
    /// <summary>
    /// Users datastore write service.
    /// </summary>
    public class UsersWriteRepository : IUsersWriteRepository
    {
        private readonly IDynamoDBContext _dynamoDbContext;
        private readonly ILogger<UsersWriteRepository> _logger;
        private readonly LambdaParameters _lambdaParameters;

        /// <summary>
        /// Creates a new instance of <see cref="UsersWriteRepository"/> class.
        /// </summary>
        /// <param name="dynamoDbContext"><see cref="DynamoDBContext"/></param>
        /// <param name="logger">Logger instance.</param>
        public UsersWriteRepository(IDynamoDBContext dynamoDbContext,
ILogger<UsersWriteRepository> logger, IOptions<LambdaParameters> lambdaParameters)
        {
            _dynamoDbContext = dynamoDbContext;
            _logger = logger;
            _lambdaParameters = lambdaParameters.Value;
        }

        /// <inheritdoc/>
        public Task<UserDto> SaveAsync(UserDto user)
        {
            if (user == null)
            {
                throw new ArgumentNullException(nameof(user));
            }

            return SaveInternalAsync(user);
        }

        /// <inheritdoc/>
        public Task<UserDto> DeleteAsync(Guid id)
        {
            if (string.IsNullOrEmpty(id.ToString()))
            {
                throw new ArgumentNullException(nameof(id));
            }

            return DeleteUserInternalAsync(id);
        }

        private async Task<UserDto> SaveInternalAsync(UserDto user)
        {
            _logger.LogDebug("Trying to save user entity: {@user}", user);

            var config = new DynamoDBOperationConfig
            {
                OverrideTableName = _lambdaParameters.UsersTableName,
                IgnoreNullValues = false
            };

            await _dynamoDbContext.SaveAsync(user, config);
            _logger.LogInformation("Successfully created a new user entity:
{@user}", user);

            return user;
        }

        private async Task<UserDto> DeleteUserInternalAsync(Guid id)
        {
            _logger.LogDebug("Trying to delete user entity with id: {id}", id);
```

```
            var config = new DynamoDBOperationConfig
            {
                OverrideTableName = _lambdaParameters.UsersTableName,
            };

            UserDto user = await _dynamoDbContext.LoadAsync<UserDto>(id, config);
            if (user == null)
            {
                throw new UserNotFoundException();
            }

            await _dynamoDbContext.DeleteAsync<UserDto>(id, config);
            _logger.LogInformation("Successfully deleted user entity: {@userDto}",
user);
            return user;
        }
    }
}
```

Файл CircuitBreakingService.cs

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.SQS.Model;
using Common.Interfaces;
using FluentValidation;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using Resiliency.CircuitBreaker;
using System.Net;
using UserIntegrationLambda.Interfaces.CircuitBreaker;
using UserIntegrationLambda.Options;
using static Amazon.Lambda.SQSEvents.SQSEvent;

namespace UserIntegrationLambda.Services.CircuitBreaker
{
    /// <summary>
    /// Circuit Breaker service adapter.
    /// </summary>
    public class CircuitBreakingService : ICircuitBreakingService
    {
        private readonly ICircuitStateRepository _circuitStateRepository;
        private readonly IEventBridgeSchedulerService
_circuitBreakerClosureScheduler;
        private readonly IEventSourceMappingClient _eventSourceMappingClient;
        private readonly CircuitBreakerOptions _circuitBreakerOptions;
        private readonly ISqsClient _sqsClient;
        private readonly ILogger<CircuitBreakingService> _logger;

        /// <summary>
        /// Initializes a new instance of <see cref="CircuitBreakingService"/>
class.
        /// </summary>
        /// <param name="circuitStateRepository">Circuit state datastore.</param>
        /// <param name="circuitBreakerClosureScheduler">Service to work with Event
Bridge Schedule.</param>
        /// <param name="eventSourceMappingClient">Service to manage lambda event
source.</param>
        /// <param name="circuitBreakerOptions"><see
cref="CircuitBreakerOptions"/></param>
        /// <param name="sqsClient"></param>
        /// <param name="logger">Logger instance.</param>
```

```
        public CircuitBreakingService(ICircuitStateRepository
circuitStateRepository,
            IEventBridgeSchedulerService circuitBreakerClosureScheduler,
            IEventSourceMappingClient eventSourceMappingClient,
            IOptions<CircuitBreakerOptions> circuitBreakerOptions,
            ISqsClient sqsClient,
            ILogger<CircuitBreakingService> logger)
        {
            _circuitStateRepository = circuitStateRepository;
            _circuitBreakerClosureScheduler = circuitBreakerClosureScheduler;
            _eventSourceMappingClient = eventSourceMappingClient;
            _circuitBreakerOptions = circuitBreakerOptions.Value;
            _sqsClient = sqsClient;
            _logger = logger;
        }

        /// <inheritdoc/>
        public async Task ExecuteAsync(SQSMessage sqsMessage, Func<SQSMessage, Task>
action)
        {
            CircuitState currentState = await _circuitStateRepository.GetAsync();
            CircuitBreakerPolicy circuitBreakerPolicy = new(currentState);

            try
            {
                await circuitBreakerPolicy
                    .Handle(IsTransientError)
                    .ExecuteAsync(() => action(sqsMessage));
            }
            catch (OpenCircuitException openCircuitException)
            {
                await _eventSourceMappingClient.DisableEventSourceMappingAsync();
                await
_circuitBreakerClosureScheduler.ScheduleCircuitClosureTrialAsync(openCircuitExceptio
n.OpenUntil);
                await _sqsClient.EnqueueAsync(sqsMessage,
_circuitBreakerOptions.SourceQueueUrl);
            }
            catch (Exception ex) when (IsPermanentError(ex))
            {
                _logger.LogError(ex, "Failed to handle SQS message due to permanent
error");
                await _sqsClient.EnqueueAsync(sqsMessage,
_circuitBreakerOptions.DeadLetterQueueUrl);
            }
            finally
            {
                await _circuitStateRepository.SaveAsync(circuitBreakerPolicy.State);
            }
        }

        /// <inheritdoc/>
        public async Task Close()
        {
            _logger.LogDebug("Switching the CB state to closed.");
            await _eventSourceMappingClient.EnableEventSourceMappingAsync();
            await _circuitBreakerClosureScheduler.CancelCircuitClosureTrialAsync();
            OpenStateConfig openStateConfig = new(_circuitBreakerOptions.Timeout);
            ClosedStateConfig closedStateConfig =
new(_circuitBreakerOptions.ErrorThreshold, _circuitBreakerOptions.SamplingDuration);
            var circuitState = new ClosedState(openStateConfig, closedStateConfig);
            await _circuitStateRepository.SaveAsync(circuitState);
        }
```

```
        /// <inheritdoc/>
        public async Task HalfOpen()
        {
            _logger.LogDebug("Switching the CB state to half-open.");
            await _eventSourceMappingClient.EnableEventSourceMappingAsync();
            await _circuitBreakerClosureScheduler.CancelCircuitClosureTrialAsync();
            OpenStateConfig openStateConfig = new(_circuitBreakerOptions.Timeout);
            ClosedStateConfig closedStateConfig =
new(_circuitBreakerOptions.ErrorThreshold, _circuitBreakerOptions.SamplingDuration);
            var circuitState = new HalfOpenState(openStateConfig,
closedStateConfig);
            await _circuitStateRepository.SaveAsync(circuitState);
        }

        /// <inheritdoc/>
        public async Task Open(TimeSpan? customTimeout)
        {
            _logger.LogDebug("Switching the CB state to open.");
            await _eventSourceMappingClient.EnableEventSourceMappingAsync();
            TimeSpan timeout = customTimeout ?? _circuitBreakerOptions.Timeout;
            await
_circuitBreakerClosureScheduler.ScheduleCircuitClosureTrialAsync(DateTimeOffset.UtcN
ow + timeout);
            OpenStateConfig openStateConfig = new(timeout);
            ClosedStateConfig closedStateConfig =
new(_circuitBreakerOptions.ErrorThreshold, _circuitBreakerOptions.SamplingDuration);
            OpenState circuitState = new(openStateConfig, closedStateConfig);
            await _circuitStateRepository.SaveAsync(circuitState);
        }

        /// <inheritdoc/>
        public async Task PermanentlyClose()
        {
            _logger.LogDebug("Switching the CB state to permanently closed.");
            await _eventSourceMappingClient.EnableEventSourceMappingAsync();
            await _circuitBreakerClosureScheduler.CancelCircuitClosureTrialAsync();
            var circuitState = new PermanentlyClosedState();
            await _circuitStateRepository.SaveAsync(circuitState);
        }

        /// <inheritdoc/>
        public async Task Trial(Func<SQSMessage, Task> action)
        {
            _logger.LogDebug("Starting Circuit Breaker trial.");

            Message? message = await
_sqsClient.DequeueAsync(_circuitBreakerOptions.SourceQueueUrl);
            _logger.LogDebug("Retrieved the message: {@message} from the source
queue", message);

            if (message is not null)
            {
                var sqsMessage = new SQSMessage
                {
                    Attributes = message.Attributes,
                    Body = message.Body,
                    MessageId = message.MessageId,
                    ReceiptHandle = message.ReceiptHandle
                };

                try
                {
                    await ExecuteAsync(sqsMessage, action);
```

```
                    await _sqsClient.DeleteMessageAsync(message.ReceiptHandle,
_circuitBreakerOptions.SourceQueueUrl);

                    await _eventSourceMappingClient.EnableEventSourceMappingAsync();
                    await
_circuitBreakerClosureScheduler.CancelCircuitClosureTrialAsync();
                }
                catch (Exception ex)
                {
                    _logger.LogError(ex, "Circuit Breaker Trial has failed.");
                    await
_circuitBreakerClosureScheduler.ScheduleCircuitClosureTrialAsync(DateTimeOffset.UtcN
ow + _circuitBreakerOptions.Timeout);
                }
            }
            else
            {
                _logger.LogInformation("No messages found in the source queue.
Switching the state to half-open.");
                await HalfOpen();
            }
        }

        private static bool IsTransientError(Exception exception)
        {
            var transientErrorCodes = new HashSet<HttpStatusCode>
            {
                HttpStatusCode.NotFound,
                HttpStatusCode.RequestTimeout,
                HttpStatusCode.TooManyRequests,
                HttpStatusCode.BadGateway,
                HttpStatusCode.ServiceUnavailable,
                HttpStatusCode.GatewayTimeout
            };

            return exception is ResourceNotFoundException or
AmazonDynamoDBException;
        }

        private static bool IsPermanentError(Exception exception)
        {
            var permanentErrorCodes = new HashSet<HttpStatusCode>
            {
                HttpStatusCode.BadRequest,
                HttpStatusCode.Forbidden,
                HttpStatusCode.Unauthorized,
                HttpStatusCode.MethodNotAllowed,
                HttpStatusCode.Conflict,
                HttpStatusCode.RequestEntityTooLarge
            };

            return exception is ValidationException;
        }
    }
}
```

Файл CircuitStateToDatabaseDtoMapper.cs

```
using Resiliency.CircuitBreaker;
using UserIntegrationLambda.Interfaces.CircuitBreaker;
using UserIntegrationLambda.Models;

namespace UserIntegrationLambda.Services.CircuitBreaker
```

```
{
    /// <summary>
    /// Mapper class for Circuit State Database DTO.
    /// </summary>
    public class CircuitStateToDatabaseDtoMapper : ICircuitStateToDatabaseDtoMapper
    {
        /// <inheritdoc/>
        public CircuitStateDatabaseDto? CircuitStateDatabaseDto { get; private set;
}

        /// <inheritdoc/>
        public void Visit(OpenState state)
        {
            CircuitStateDatabaseDto = new CircuitStateDatabaseDto
            {
                StateId = "Open",
                OpenedAt = state.OpenedAt.DateTime
            };
        }

        /// <inheritdoc/>
        public void Visit(ClosedState state)
        {
            CircuitStateDatabaseDto = new CircuitStateDatabaseDto
            {
                StateId = "Closed",
                ErrorCounter = state.ErrorCounter,
                StartedAt = state.StartedAt.DateTime
            };
        }

        /// <inheritdoc/>
        public void Visit(HalfOpenState state)
        {
            CircuitStateDatabaseDto = new CircuitStateDatabaseDto
            {
                StateId = "HalfOpen"
            };
        }

        /// <inheritdoc/>
        public void Visit(PermanentlyClosedState state)
        {
            CircuitStateDatabaseDto = new CircuitStateDatabaseDto
            {
                StateId = "PermanentlyClosed",
                StartedAt = state.StartedAt.DateTime
            };
        }
    }
}
```

Файл SqsEventSourceMappingClient.cs

```
using Amazon.Lambda;
using Amazon.Lambda.Model;
using Common.Entities;
using Common.Interfaces;
using Common.Exceptions;
using Microsoft.Extensions.Logging;
using System.Net;

namespace UserIntegrationLambda.Services.CircuitBreaker
{
    /// <summary>
```

```
    /// SQS event source mapping switcher (on/off).
    /// </summary>
    public class SqsEventSourceMappingClient : IEventSourceMappingClient
    {
        private readonly IAmazonLambda _amazonLambdaClient;
        private readonly ILambdaContextAccessor _lambdaContextAccessor;
        private readonly ILogger<SqsEventSourceMappingClient> _logger;

        /// <summary>
        /// Initializes a new instance of <see cref="SqsEventSourceMappingClient"/>
class.
        /// </summary>
        /// <param name="amazonLambdaClient"><see cref="IAmazonLambda"/></param>
        /// <param name="lambdaContextAccessor"><see
cref="ILambdaContextAccessor"/></param>
        /// <param name="logger">Logger instance.</param>
        public SqsEventSourceMappingClient(IAmazonLambda amazonLambdaClient,
ILambdaContextAccessor lambdaContextAccessor, ILogger<SqsEventSourceMappingClient>
logger)
        {
            _amazonLambdaClient = amazonLambdaClient;
            _lambdaContextAccessor = lambdaContextAccessor;
            _logger = logger;
        }

        /// <inheritdoc/>
        public async Task DisableEventSourceMappingAsync()
        {
            _logger.LogDebug("Trying to disable EventSourceMapping for
{FunctionName} function", _lambdaContextAccessor.Context.FunctionName);
            EventSourceMappingConfiguration currentState = await
GetMappingConfiguration(_lambdaContextAccessor.Context.FunctionName);

            if (currentState.State == AwsEventSourceMappingState.Enabling)
            {
                throw new MappingStateTransitionException($"EventSourceMapping can't
be disabled as it's in {currentState.State} state.");
            }

            if (currentState.State is not (AwsEventSourceMappingState.Disabling or
AwsEventSourceMappingState.Disabled))
            {
                await SwitchEventSourceMappingAsync(currentState, false);
            }
        }

        /// <inheritdoc/>
        public async Task EnableEventSourceMappingAsync()
        {
            _logger.LogDebug("Trying to enable EventSourceMapping for {FunctionName}
function", _lambdaContextAccessor.Context.FunctionName);
            EventSourceMappingConfiguration currentState = await
GetMappingConfiguration(_lambdaContextAccessor.Context.FunctionName);

            if (currentState.State == AwsEventSourceMappingState.Disabling)
            {
                throw new MappingStateTransitionException($"EventSourceMapping can't
be enabled as it's in {currentState.State} state.");
            }

            if (currentState.State is not (AwsEventSourceMappingState.Enabling or
AwsEventSourceMappingState.Enabled))
            {
                await SwitchEventSourceMappingAsync(currentState, true);
```

```
            }
        }

        private async Task
SwitchEventSourceMappingAsync(EventSourceMappingConfiguration currentState, bool
enable)
        {
            if (currentState == null) throw new
ArgumentNullException(nameof(currentState));

            UpdateEventSourceMappingResponse updateResponse =
                await _amazonLambdaClient.UpdateEventSourceMappingAsync(new
UpdateEventSourceMappingRequest
                {
                    UUID = currentState.UUID,
                    Enabled = enable,
                });

            string targetState = enable ? "enable" : "disable";
            if (updateResponse.HttpStatusCode == HttpStatusCode.Accepted)
            {
                _logger.LogInformation("Request to {targetState} {arn}
EventSourceMapping is accepted", targetState, currentState.EventSourceArn);
            }
            else
            {
                throw new MappingStateTransitionException($"Request to {targetState}
{currentState.EventSourceArn} EventSourceMapping responded with
{updateResponse.HttpStatusCode}");
            }
        }

        private async Task<EventSourceMappingConfiguration>
GetMappingConfiguration(string functionName)
        {
            ListEventSourceMappingsResponse listEventSourceMappings =
                await _amazonLambdaClient.ListEventSourceMappingsAsync(new
ListEventSourceMappingsRequest { FunctionName = functionName });
            EventSourceMappingConfiguration? mappingConfiguration =
                listEventSourceMappings.EventSourceMappings.FirstOrDefault(m =>
SqsEventSourceArn.IsMatch(m.EventSourceArn));
            return mappingConfiguration ?? throw new
MappingStateTransitionException($"Failed to find SQS EventSourceMapping for
{functionName}");
        }
    }
}
```

Файл SqsEventProcessingService.cs

```
using Common.Interfaces;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json.Linq;
using UserIntegrationLambda.Interfaces;

namespace UserIntegrationLambda.Services
{
    /// <summary>
    /// SQS Event processor.
    /// </summary>
    internal class SqsEventProcessingService : ISqsEventProcessingService
    {
        private readonly IDataHandlerStrategySelector _strategySelector;
```

```
        private readonly ILogger<SqsEventProcessingService> _logger;

        /// <summary>
        /// Initializes a new instance of <see cref="SqsEventProcessingService"/>
class.
        /// </summary>
        /// <param name="strategySelector"><see
cref="IDataHandlerStrategySelector"/></param>
        /// <param name="logger">Logger instance.</param>
        public SqsEventProcessingService(IDataHandlerStrategySelector
strategySelector, ILogger<SqsEventProcessingService> logger)
        {
            _strategySelector = strategySelector;
            _logger = logger;
        }

        /// <inheritdoc/>
        public Task ProcessAsync(JObject input)
        {
            if (input is null)
            {
                throw new ArgumentNullException(nameof(input));
            }

            return ProcessInternalAsync(input);
        }

        private async Task ProcessInternalAsync(JObject input)
        {
            _logger.LogDebug("SQS message processing started. Message: {message}",
input.ToString());

            IDataHandlerStrategy dataHandler = _strategySelector.GetStrategy(input);

            await dataHandler.ProcessInputAsync(input);
        }
    }
}
```

Файл SqsRecordProcessingService.cs

```
using Amazon.Lambda.SQSEvents;
using Common.Entities;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;
using UserIntegrationLambda.Interfaces;
using UserIntegrationLambda.Interfaces.CircuitBreaker;
using static Amazon.Lambda.SQSEvents.SQSEvent;

namespace UserIntegrationLambda.Services
{
    /// <summary>
    /// SQS record processing service.
    /// </summary>
    public class SqsRecordProcessingService : ISqsRecordProcessingService
    {
        private readonly ICircuitBreakingService _circuitBreakingService;
        private readonly IUserIntegrationHandler _userIntegrationHandler;
        private readonly ILogger<SqsRecordProcessingService> _logger;

        /// <summary>
        /// Initializes a new instance of <see cref="SqsRecordProcessingService"/>
class.
```

```
        /// </summary>
        /// <param name="circuitBreakingService"></param>
        /// <param name="userIntegrationHandler"></param>
        /// <param name="logger">Logger instance.</param>
        public SqsRecordProcessingService(ICircuitBreakingService
circuitBreakingService, IUserIntegrationHandler userIntegrationHandler,
ILogger<SqsRecordProcessingService> logger)
        {
            _circuitBreakingService = circuitBreakingService;
            _userIntegrationHandler = userIntegrationHandler;
            _logger = logger;
        }

        /// <inheritdoc/>
        public async Task ProcessMessageAsync(SQSMessage sqsMessage)
        {
            if (sqsMessage is null) throw new
ArgumentNullException(nameof(sqsMessage));

            BaseMessage<UserDto>? messageBody =
JsonConvert.DeserializeObject<BaseMessage<UserDto>>(sqsMessage.Body);
            _logger.LogDebug("Deserialized SQS message body into: {@messageBody}",
messageBody);

            if (messageBody is null)
            {
                throw new ArgumentNullException(nameof(sqsMessage), "SQS Message
body is empty");
            }

            if (string.IsNullOrEmpty(messageBody.Action.ToString()))
            {
                throw new ArgumentException(nameof(sqsMessage), "Action is not
defined");
            }

            _logger.LogDebug("Executing {action} for User ID: {uuid}",
messageBody.Action, messageBody.EntityId);
            Dictionary<string, Func<Task>> defaultActions =
LoadDefaultActions(messageBody);

            if (!defaultActions.ContainsKey(messageBody.Action.ToString()))
            {
                string supportedActions = string.Join(',', defaultActions.Select(a
=> a.Key));
                _logger.LogWarning("Action {action} is not supported. See suported
actions: [{supportedActions}]", messageBody.Action, supportedActions);
                return;
            }

            Func<Task> function = defaultActions[messageBody.Action.ToString()];
            await function();

            _logger.LogInformation("{action} successfully executed for user:
{uuid}", messageBody.Action, messageBody.EntityId);
        }

        /// <inheritdoc/>
        public async Task ProcessSqsRecordsAsync(SQSEvent sqsEvent)
        {
            foreach (SQSMessage message in sqsEvent.Records)
            {
                _logger.LogDebug("Started processing SQSMessage {MessageId}",
message.MessageId);
```

```
                await _circuitBreakingService.ExecuteAsync(message,
ProcessMessageAsync);
                _logger.LogInformation("Processed SQSMessage {MessageId}",
message.MessageId);
            }
        }

        private Dictionary<string, Func<Task>>
LoadDefaultActions(BaseMessage<UserDto> message)
        {
            var defaultActions = new Dictionary<string, Func<Task>>()
            {
                { "Create", () =>
_userIntegrationHandler.CreateUser(message.Content) },
                { "Update", () =>
_userIntegrationHandler.UpdateUser(message.Content) },
                { "Delete", () =>
_userIntegrationHandler.DeleteUser(Guid.Parse(message.EntityId)) },
            };

            return defaultActions;
        }
    }
}
```

Файл UserIntegrationHandler.cs

```
using Common.Entities;
using FluentValidation;
using Microsoft.Extensions.Logging;
using UserIntegrationLambda.Interfaces;
using UserIntegrationLambda.Validation;

namespace UserIntegrationLambda.Services
{
    /// <summary>
    /// User synchronization service between lambda and downstream consumer
    /// </summary>
    public class UserIntegrationHandler : IUserIntegrationHandler
    {
        private readonly UserValidator _userValidator = new();

        private readonly IUsersWriteRepository _usersWriteRepository;
        private readonly ILogger<UserIntegrationHandler> _logger;

        /// <summary>
        /// Initializes a new instance of <see cref="UserIntegrationHandler"/>
class.
        /// </summary>
        /// <param name="usersWriteRepository">Performs write operations with the
datastore.</param>
        /// <param name="logger">Logger instance.</param>
        public UserIntegrationHandler(IUsersWriteRepository usersWriteRepository,
ILogger<UserIntegrationHandler> logger)
        {
            _usersWriteRepository = usersWriteRepository;
            _logger = logger;
        }

        /// <inheritdoc/>
        public async Task CreateUser(UserDto user)
        {
            _userValidator.ValidateAndThrow(user);
            await _usersWriteRepository.SaveAsync(user);
        }
```

```
        /// <inheritdoc/>
        public async Task DeleteUser(Guid uuid)
        {
            await _usersWriteRepository.DeleteAsync(uuid);
        }


        /// <inheritdoc/>
        public async Task UpdateUser(UserDto user)
        {
            _userValidator.ValidateAndThrow(user);
            await _usersWriteRepository.SaveAsync(user);
        }
    }
}
```

Файл UserValidator.cs

```
using Common.Constants;
using Common.Entities;
using FluentValidation;

namespace UserIntegrationLambda.Validation
{
    public class UserValidator : AbstractValidator<UserDto>
    {
        public UserValidator()
        {
            RuleFor(x => x.Uuid.ToString()).Cascade(CascadeMode.Stop)
                .NotEmpty().WithMessage("Uuid should not be empty")
                .Matches(ValidationConstants.GuidRegexPattern).WithMessage("Uuid
should of guid type");
            RuleFor(x => x.FirstName)
                .NotEmpty().WithMessage("First name should not be empty");

        }
    }
}
```

Файл Function.cs

```
using Amazon.Lambda.Core;
using Amazon.Lambda.Serialization.Json;
using Common.Interfaces;
using Common.Utils;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json.Linq;
using System.Diagnostics.CodeAnalysis;
using UserIntegrationLambda.Builders;


// Assembly attribute to enable the Lambda function's JSON input to be converted
into a .NET class.
[assembly: LambdaSerializer(typeof(JsonSerializer))]
namespace UserIntegrationLambda;

/// <summary>
/// Lambda function class.
/// </summary>
[ExcludeFromCodeCoverage]
public class Function
```

```
{
    private readonly IServiceProvider _serviceProvider;

    /// <summary>
    /// Default constructor. This constructor is used by Lambda to construct the
instance. When invoked in a Lambda environment
    /// the AWS credentials will come from the IAM role associated with the function
and the AWS region will be set to the
    /// region the Lambda function is executed in.
    /// </summary>
    public Function()
    {
        var services = new ServiceCollection();
        _serviceProvider = new ServiceProviderBuilder().Build(services);
    }

    /// <summary>
    /// This method is called for every Lambda invocation. This method takes in an
SQS event object and can be used
    /// to respond to SQS messages.
    /// </summary>
    /// <param name="evnt"></param>
    /// <param name="context"></param>
    /// <returns></returns>
    public async Task FunctionHandler(JObject input, ILambdaContext context)
    {
        ArgumentNullException.ThrowIfNull(input);
        ArgumentNullException.ThrowIfNull(context);

        var logger = _serviceProvider.GetRequiredService<ILogger<Function>>();

        try
        {
            var lambdaContextAccessor =
_serviceProvider.GetRequiredService<LambdaContextAccessor>();
            lambdaContextAccessor.Context = context;

            logger.LogInformation("Received SQS Event: {evnt}", input.ToString());

            var sqsEventProcessingService =
_serviceProvider.GetRequiredService<ISqsEventProcessingService>();

            await sqsEventProcessingService.ProcessAsync(input);
        }
        catch (Exception ex)
        {
            logger.LogCritical(ex, "An unhandled error has occured. Check the
exception details for further details.");
            throw;
        }

    }
}
```

Файл UsersController.cs

```
using Common.Entities;
using Microsoft.AspNetCore.Mvc;
using Swashbuckle.AspNetCore.Annotations;
using System.ComponentModel.DataAnnotations;
using UserManagementLambda.Interfaces;

namespace UserManagementLambda.Controllers;

/// <summary>
```

```
/// API controller for users management in License Management Service.
/// </summary>
[ApiController]
[Route("users-api/users")]
[SwaggerTag("Controller for users management")]
public class UsersController : ControllerBase
{
    private readonly IUserManagementService _userManagementService;
    private readonly ILogger<UsersController> _logger;

    /// <summary>
    /// Initializes a new instance of <see cref="UsersController"/> class.
    /// </summary>
    /// <param name="userManagementService"><see
cref="IUserManagementService"/></param>
    /// <param name="logger">Logger instance.</param>
    public UsersController(IUserManagementService userManagementService,
ILogger<UsersController> logger)
    {
        _userManagementService = userManagementService;
        _logger = logger;
    }

    /// <summary>
    /// Get user with a specific Uuid.
    /// </summary>
    /// <param name="id">User's unique identifier.</param>
    /// <returns><see cref="IActionResult"/></returns>
    [HttpGet]
    public async Task<IActionResult> GetUser([Required, FromQuery] Guid id)
    {
        UserDto user = await _userManagementService.GetUserByUuid(id);

        return Ok(user);
    }

    /// <summary>
    /// Creates a new user.
    /// </summary>
    /// <param name="user"><see cref="UserDto"/></param>
    /// <returns><see cref="IActionResult"/></returns>
    [HttpPost]
    public async Task<IActionResult> CreateUser([FromBody, Required] UserDto user)
    {
        var createdUser = await _userManagementService.CreateUser(user);

        return Accepted(createdUser);
    }

    /// <summary>
    /// Updates a specific user.
    /// </summary>
    /// <param name="user"><see cref="UserDto"/></param>
    /// <returns><see cref="IActionResult"/></returns>
    [HttpPut]
    public async Task<IActionResult> UpdateUser([FromBody, Required] UserDto user)
    {
        var updatedUser = await _userManagementService.UpdateUser(user);

        return Accepted(updatedUser);
    }

    /// <summary>
    /// Deletes a user with specific Uuid.
```

```
    /// </summary>
    /// <param name="id">User's unique indentifier.</param>
    /// <returns><see cref="IActionResult"/></returns>
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteUser(Guid id)
    {
        await _userManagementService.DeleteUser(id);

        return Accepted();
    }
}
```

Файл IUserManagementService.cs

```
using Common.Entities;

namespace UserManagementLambda.Interfaces
{
    /// <summary>
    /// Interface of user synchronization adapter.
    /// </summary>
    public interface IUserManagementService
    {
        /// <summary>
        /// Gets user by uuid from the datastore.
        /// </summary>
        /// <param name="uuid">User's ID.</param>
        /// <returns></returns>
        Task<UserDto> GetUserByUuid(Guid uuid);

        /// <summary>
        /// Gets user by username from the datastore.
        /// </summary>
        /// <param name="userName">User's name.</param>
        /// <returns></returns>
        Task<UserDto> GetUserByUserName(string userName);

        /// <summary>
        /// Sends user creation request to the SNS topic.
        /// </summary>
        /// <param name="user"><see cref="UserDto"/></param>
        /// <returns></returns>
        Task<UserDto> CreateUser(UserDto user);

        /// <summary>
        /// Sends user updation request to the SNS topic.
        /// </summary>
        /// <param name="user"><see cref="UserDto"/></param>
        /// <returns></returns>
        Task<UserDto> UpdateUser(UserDto user);

        /// <summary>
        /// Sends user deletion request to the SNS topic.
        /// </summary>
        /// <param name="uuid">User's ID.</param>
        /// <returns></returns>
        Task DeleteUser(Guid uuid);
    }
}
```

Файл IUsersReadRepository.cs

```
using Common.Entities;
```

```
using Common.Interfaces;

namespace UserManagementLambda.Interfaces
{
    /// <summary>
    /// Interface of Users datastore read service.
    /// </summary>
    public interface IUsersReadRepository : IReadRepository<UserDto>
    {
        /// <summary>
        /// Gets user by username.
        /// </summary>
        /// <param name="username">Username.</param>
        /// <returns>User dto.</returns>
        Task<UserDto> GetByUsernameAsync(string username);
    }
}
```

Файл LambdaParameters.cs

```
namespace UserManagementLambda.Options
{
    /// <summary>
    /// Contains lambda parameters.
    /// </summary>
    public class LambdaParameters
    {
        /// <summary>
        /// Amazon Resource Number for Simple Notification Service.
        /// </summary>
        public string SnsTopicArn { get; set; } = null!;
    }
}
```

Файл UsersReadRepository.cs

```
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Common.Entities;
using Common.Exceptions;
using UserManagementLambda.Interfaces;

namespace UserManagementLambda.Repositories
{
    /// <summary>
    /// Repository that represents user entity datastore.
    /// </summary>
    public class UsersReadRepository : IUsersReadRepository
    {
        private readonly IDynamoDBContext _dynamoDbContext;
        private readonly ILogger<UsersReadRepository> _logger;

        /// <summary>
        /// Creates a new instance of <see cref="UsersReadRepository"/> class.
        /// </summary>
        /// <param name="dynamoDbContext"><see cref="DynamoDBContext"/></param>
        /// <param name="logger">Logger instance.</param>
        public UsersReadRepository(IDynamoDBContext dynamoDbContext,
ILogger<UsersReadRepository> logger)
        {
            _dynamoDbContext = dynamoDbContext;
```

```
            _logger = logger;
        }

        /// <inheritdoc/>
        public Task<UserDto> GetByIdAsync(Guid id)
        {
            if (string.IsNullOrEmpty(id.ToString()))
            {
                throw new ArgumentNullException(nameof(id));
            }

            return GetUserByIdInternalAsync(id);
        }

        /// <inheritdoc/>
        public Task<UserDto> GetByUsernameAsync(string username)
        {
            if (string.IsNullOrEmpty(username))
            {
                throw new ArgumentNullException(nameof(username));
            }

            return GetUserByUsernameInternalAsync(username);
        }

        private async Task<UserDto> GetUserByIdInternalAsync(Guid id)
        {
            _logger.LogDebug("Trying to get user entity with id: {id}", id);

            UserDto user = await _dynamoDbContext.LoadAsync<UserDto>(id);

            _logger.LogInformation("Successfully retrieved a new user entity:
{@userDto}", user);

            return user;
        }

        private async Task<UserDto> GetUserByUsernameInternalAsync(string username)
        {
            _logger.LogDebug("Trying to get user entity with username: {username}",
username);

            var scanCondition = new ScanCondition("Username", ScanOperator.Equal,
username);
            var searchResult = await _dynamoDbContext.ScanAsync<UserDto>(new[] {
scanCondition })
                .GetRemainingAsync();

            if (searchResult is null || searchResult.FirstOrDefault() is null)
            {
                throw new UserNotFoundException();
            }

            _logger.LogInformation("Successfully retrieved a new user entity:
{@userDto}", searchResult);

            return searchResult.First();
        }
    }
}
```

Файл UserManagementService.cs

```
using Common.Constants;
using Common.Entities;
using Common.Exceptions;
using Common.Interfaces;
using Microsoft.Extensions.Options;
using UserManagementLambda.Interfaces;
using UserManagementLambda.Options;

namespace UserManagementLambda.Services
{
    /// <summary>
    /// User synchronization adapter between service and SNS.
    /// </summary>
    public class UserManagementService : IUserManagementService
    {
        private readonly IUsersReadRepository _usersRepository;
        private readonly ISnsClient _snsService;
        private readonly LambdaParameters _environmentVariables;
        private readonly ILogger<UserManagementService> _logger;

        /// <summary>
        /// Initializes a new instance of <see cref="UserManagementService"/> class.
        /// </summary>
        /// <param name="usersRepository"><see cref="IUsersReadRepository"/></param>
        /// <param name="snsService"><see cref="ISnsClient"/></param>
        /// <param name="environmentVariables">Lambda environment variables.</param>
        /// <param name="logger">Logger instance.</param>
        public UserManagementService(IUsersReadRepository usersRepository,
ISnsClient snsService, IOptions<LambdaParameters> environmentVariables,
ILogger<UserManagementService> logger)
        {
            _usersRepository = usersRepository;
            _snsService = snsService;
            _environmentVariables = environmentVariables.Value;
            _logger = logger;
        }

        /// <inheritdoc/>
        public async Task<UserDto> CreateUser(UserDto user)
        {
            var userMessage = new BaseMessage<UserDto>(user.Uuid.ToString(),
EntityTypes.User, ProcessAction.Create)
            {
                Content = user
            };

            await _snsService.PublishToTopicAsync(_environmentVariables.SnsTopicArn,
userMessage);

            return user;
        }

        /// <inheritdoc/>
        public async Task DeleteUser(Guid uuid)
        {
            var userMessage = new BaseMessage<UserDto>(uuid.ToString(),
EntityTypes.User, ProcessAction.Delete);

            await _snsService.PublishToTopicAsync(_environmentVariables.SnsTopicArn,
userMessage);
        }

        /// <inheritdoc/>
        public async Task<UserDto> GetUserByUserName(string userName)
```

```
        {
            var user = await _usersRepository.GetByUsernameAsync(userName);

            return user;
        }

        /// <inheritdoc/>
        public async Task<UserDto> GetUserByUuid(Guid uuid)
        {
            var user = await _usersRepository.GetByIdAsync(uuid);

            if (user is null)
            {
                throw new UserNotFoundException();
            }

            return user;
        }

        /// <inheritdoc/>
        public async Task<UserDto> UpdateUser(UserDto user)
        {
            var existingUser = await _usersRepository.GetByIdAsync(user.Uuid);

            if (existingUser is null) throw new UserNotFoundException();

            var userMessage = new BaseMessage<UserDto>(user.Uuid.ToString(),
EntityTypes.User, ProcessAction.Update)
            {
                Content = user
            };

            await _snsService.PublishToTopicAsync(_environmentVariables.SnsTopicArn,
userMessage);

            return user;
        }
    }
}
```

Файл Startup.cs

```
using Common.Extensions;
using Common.Interfaces;
using Common.Middleware;
using Common.Services;
using Microsoft.OpenApi.Models;
using System.Diagnostics.CodeAnalysis;
using UserManagementLambda.Interfaces;
using UserManagementLambda.Options;
using UserManagementLambda.Repositories;
using UserManagementLambda.Services;

namespace UserManagementLambda;

/// <summary>
/// Startup class.
/// </summary>
[ExcludeFromCodeCoverage]
public class Startup
{
    private readonly IConfiguration _configuration;
```

```
    /// <summary>
    /// Intializes a new instance of <see cref="Startup"/> class.
    /// </summary>
    /// <param name="configuration"><see cref="IConfiguration"/></param>
    /// <exception cref="ArgumentNullException"></exception>
    public Startup(IConfiguration configuration)
    {
        _configuration = configuration ?? throw new
ArgumentNullException(nameof(configuration));
    }

    /// <summary>
    /// Adds services to the DI container. This method gets called by the runtime.
    /// </summary>
    /// <param name="services"><see cref="IServiceCollection"/></param>
    public void ConfigureServices(IServiceCollection services)
    {
        services.ConfigureLambdaVariables<LambdaParameters>(_configuration);

        services.ConfigureLogging();

        services.ConfigureDynamoDB(_configuration);
        services.ConfigureSns();

        services.AddControllers();

        services.ConfigureSwaggerServices(new OpenApiInfo
        {
            Version = "v1",
            Title = "User Management Lambda",
            Description = "Users API Lambda implementation for License Management
Service."
        });

        services.AddScoped<ISnsClient, SnsClient>();
        services.AddScoped<IUsersReadRepository, UsersReadRepository>();
        services.AddScoped<IUserManagementService, UserManagementService>();
    }

    /// <summary>
    /// Adds services to the DI container. This method gets called by the runtime.
    /// </summary>
    /// <param name="services"><see cref="IServiceCollection"/></param>
    public void Configure(IApplicationBuilder app)
    {
        app.UseHttpsRedirection();

        app.UseRouting();

        app.UseMiddleware<UnhandledExceptionLoggingMiddleware>();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllerRoute(
                name: "default",
                pattern: "users-api/{controller}/{id?}");
        });

        app.UseSwagger("users-api", _configuration);
    }
}
```

Файл api_gateway main.tf

```
resource "aws_api_gateway_rest_api" "lambda_api" {
    name        = var.name
    description = "Automated deployment of API Gateway for Lambda"
}

# Users API endpoint
resource "aws_api_gateway_resource" "users-api" {
    rest_api_id = aws_api_gateway_rest_api.lambda_api.id
    parent_id   = aws_api_gateway_rest_api.lambda_api.root_resource_id
    path_part   = "users-api"
}

resource "aws_api_gateway_method" "users-api-get-method" {
    rest_api_id   = aws_api_gateway_rest_api.lambda_api.id
    resource_id   = aws_api_gateway_resource.users-api.id
    http_method   = "GET"
    authorization = "NONE"
}

resource "aws_api_gateway_integration" "users-api-root-get-integration" {
    rest_api_id = aws_api_gateway_rest_api.lambda_api.id
    resource_id = aws_api_gateway_method.users-api-get-method.resource_id
    http_method = aws_api_gateway_method.users-api-get-method.http_method

    integration_http_method = "POST"
    type                    = "AWS_PROXY"
    uri                     = var.users_uri
}

resource "aws_api_gateway_resource" "users-api-proxy" {
    rest_api_id = aws_api_gateway_rest_api.lambda_api.id
    parent_id   = aws_api_gateway_resource.users-api.id
    path_part   = "{proxy+}"
}

resource "aws_api_gateway_method" "users-api-proxy-method" {
    rest_api_id   = aws_api_gateway_rest_api.lambda_api.id
    resource_id   = aws_api_gateway_resource.users-api-proxy.id
    http_method   = "ANY"
```

```
    authorization = "NONE"
}

resource "aws_api_gateway_integration" "users-api-integration" {
    rest_api_id = aws_api_gateway_rest_api.lambda_api.id
    resource_id = aws_api_gateway_method.users-api-proxy-method.resource_id
    http_method = aws_api_gateway_method.users-api-proxy-method.http_method

    integration_http_method = "POST"
    type                    = "AWS_PROXY"
    uri                     = var.users_uri
}

# Products API endpoint
resource "aws_api_gateway_resource" "products-api" {
    rest_api_id = aws_api_gateway_rest_api.lambda_api.id
    parent_id   = aws_api_gateway_rest_api.lambda_api.root_resource_id
    path_part   = "products-api"
}

resource "aws_api_gateway_method" "products-api-get-method" {
    rest_api_id   = aws_api_gateway_rest_api.lambda_api.id
    resource_id   = aws_api_gateway_resource.products-api.id
    http_method   = "GET"
    authorization = "NONE"
}

resource "aws_api_gateway_integration" "products-api-root-get-integration" {
    rest_api_id = aws_api_gateway_rest_api.lambda_api.id
    resource_id = aws_api_gateway_method.products-api-get-method.resource_id
    http_method = aws_api_gateway_method.products-api-get-method.http_method

    integration_http_method = "POST"
    type                    = "AWS_PROXY"
    uri                     = var.products_uri
}

resource "aws_api_gateway_resource" "products-api-proxy" {
    rest_api_id = aws_api_gateway_rest_api.lambda_api.id
```

```
    parent_id   = aws_api_gateway_resource.products-api.id
    path_part   = "{proxy+}"
}

resource "aws_api_gateway_method" "products-api-proxy-method" {
    rest_api_id   = aws_api_gateway_rest_api.lambda_api.id
    resource_id   = aws_api_gateway_resource.products-api-proxy.id
    http_method   = "ANY"
    authorization = "NONE"
}

resource "aws_api_gateway_integration" "products-api-integration" {
    rest_api_id = aws_api_gateway_rest_api.lambda_api.id
    resource_id = aws_api_gateway_method.products-api-proxy-method.resource_id
    http_method = aws_api_gateway_method.products-api-proxy-method.http_method

    integration_http_method = "POST"
    type                    = "AWS_PROXY"
    uri                     = var.products_uri
}

#License API endpoint
resource "aws_api_gateway_resource" "license-api" {
    rest_api_id = aws_api_gateway_rest_api.lambda_api.id
    parent_id   = aws_api_gateway_rest_api.lambda_api.root_resource_id
    path_part   = "license-api"
}

resource "aws_api_gateway_method" "license-api-get-method" {
    rest_api_id   = aws_api_gateway_rest_api.lambda_api.id
    resource_id   = aws_api_gateway_resource.license-api.id
    http_method   = "GET"
    authorization = "NONE"
}

resource "aws_api_gateway_integration" "license-api-root-get-integration" {
    rest_api_id = aws_api_gateway_rest_api.lambda_api.id
    resource_id = aws_api_gateway_method.license-api-get-method.resource_id
    http_method = aws_api_gateway_method.license-api-get-method.http_method
```

```
    integration_http_method = "POST"
    type                    = "AWS_PROXY"
    uri                     = var.license_uri
}

resource "aws_api_gateway_resource" "license-api-proxy" {
    rest_api_id = aws_api_gateway_rest_api.lambda_api.id
    parent_id   = aws_api_gateway_resource.license-api.id
    path_part   = "{proxy+}"
}

resource "aws_api_gateway_method" "license-api-proxy-method" {
    rest_api_id   = aws_api_gateway_rest_api.lambda_api.id
    resource_id   = aws_api_gateway_resource.license-api-proxy.id
    http_method   = "ANY"
    authorization = "NONE"
}

resource "aws_api_gateway_integration" "license-api-integration" {
    rest_api_id = aws_api_gateway_rest_api.lambda_api.id
    resource_id = aws_api_gateway_method.license-api-proxy-method.resource_id
    http_method = aws_api_gateway_method.license-api-proxy-method.http_method

    integration_http_method = "POST"
    type                    = "AWS_PROXY"
    uri                     = var.license_uri
}

# REST API Deployment in AWS
resource "aws_api_gateway_deployment" "api_deploy" {
    depends_on = [
        aws_api_gateway_method.users-api-get-method,
        aws_api_gateway_method.users-api-proxy-method,
        aws_api_gateway_method.license-api-get-method,
        aws_api_gateway_method.license-api-proxy-method,
        aws_api_gateway_method.products-api-get-method,
        aws_api_gateway_method.products-api-proxy-method
    ]
```

```
    rest_api_id = aws_api_gateway_rest_api.lambda_api.id

    triggers = {
        redeployment = sha1(jsonencode(aws_api_gateway_rest_api.lambda_api.body))
    }

    lifecycle {
      create_before_destroy = true
    }
}

resource "aws_api_gateway_stage" "development_stage" {
    deployment_id = aws_api_gateway_deployment.api_deploy.id
    rest_api_id   = aws_api_gateway_rest_api.lambda_api.id
    stage_name    = var.stage_name
}

# Add permissions for API Gateway to call all lambdas
resource "aws_lambda_permission" "function_permission_for_apigateway" {
    for_each      = var.lambda_names

    statement_id  = "AllowAPIGatewayInvoke"
    action        = "lambda:InvokeFunction"
    function_name = each.key
    principal     = "apigateway.amazonaws.com"

    source_arn    = "${aws_api_gateway_rest_api.lambda_api.execution_arn}/*/*"
}
```

Файл api_gateway output.tf

```
output "invoke_url" {
    description = "URL to invoke API pointing to the stage"
    value       =
"${aws_api_gateway_deployment.api_deploy.invoke_url}${var.stage_name}/"
}
```

Файл api_gateway variables.tf

```
output "invoke_url" {
    description = "URL to invoke API pointing to the stage"
```

```
    value       =
"${aws_api_gateway_deployment.api_deploy.invoke_url}${var.stage_name}/"
}
```

Файл dynamodb main.tf

```
resource "aws_dynamodb_table" "dynamoDB" {
  name = var.name
  billing_mode = var.billing_mode
  hash_key = var.hash_key
  read_capacity = var.read_capacity
  write_capacity = var.write_capacity

  dynamic attribute {
    for_each = var.attributes
    content {
      name = attribute.value.name
      type = attribute.value.type
    }
  }
}
```

Файл dynamodb output.tf

```
output "arn" {
    description = "Dynamo DB table ARN"
    value       = aws_dynamodb_table.dynamoDB.arn
}
output "name" {
    description = "Dynamo DB table name"
    value       = aws_dynamodb_table.dynamoDB.id
}
```

Файл dynamodb variables.tf

```
variable "name" {
  description = "Name of the DynamoDB table"
  type = string
}

variable "hash_key" {
  description = "The attribute to use as the hash (partition) key. Must also be
defined as an attribute"
  type = string
```

```
}

variable "billing_mode" {
  description = "Controls how you are billed for read/write capacity"
  type = string
  default = "PAY_PER_REQUEST"
}

variable "read_capacity" {
  description = "The number of read units for this table"
  type = number
  default = 0
}

variable "write_capacity" {
  description = "The number of write unites for this table"
  type = number
  default = 0
}

variable "attributes" {
  description = "List of nested attribute definitions. Only required for hash_key
and range_key attributes"
  type = list(map(string))
  default = []
}
```

Файл lambda main.tf

```
resource "aws_lambda_function" "lambda_function"{
    function_name     = "${var.prefix}-${var.name}"
    description       = var.description
    filename          = var.filename
    source_code_hash  = filebase64sha256(var.filename)
    handler           = var.handler
    runtime           = var.runtime
    memory_size       = var.memory_size
    timeout           = var.timeout
    role              = var.role_arn
    publish           = var.publish
```

```
    tracing_config {
      mode          = "Active"
    }

    dynamic "environment" {
      for_each = length(var.environment_variables) > 0 ?
[var.environment_variables] : []
      content {
        variables = environment.value
      }
    }

    vpc_config {
      subnet_ids         = var.subnet_ids
      security_group_ids = var.security_group_ids
    }

    depends_on = [ aws_cloudwatch_log_group.log_group ]
}

resource "aws_lambda_alias" "lambda_alias_current" {
  name                = var.lambda_alias_current
  function_name       = aws_lambda_function.lambda_function.arn
  function_version    = aws_lambda_function.lambda_function.version
}

resource "aws_cloudwatch_log_group" "log_group" {
    name              = "/aws/lambda/${var.prefix}-${var.name}"
    retention_in_days = 365
}
```

Файл lambda output.tf

```
output "lambda_name" {
    description = "Name of the Lambda function"
    value       = aws_lambda_function.lambda_function.function_name
}

output "lambda_arn" {
```

```
    description = "Amazon Resource Name of the Lambda function"
    value       = aws_lambda_function.lambda_function.arn
}

output "lambda_invoke_arn" {
    description = "ARN used to invoke Lambda Function from API Gateway"
    value       = aws_lambda_function.lambda_function.invoke_arn
}
```

Файл lambda variables.tf

```
variable "prefix" {
  description   = "Prefix for project name"
  type          = string
}

variable "name" {
    description   = "Prefix for the name of the lambda"
    type          = string
}

variable "description" {
    description   = "Description for the lambda (displayed in AWS console)"
    type          = string
}

variable "filename" {
    description   = "The path for the function's deployment package within the
local filesystem"
    type          = string
}

variable "handler" {
    description   = "Name of the handler function in the code"
    type          = string
}

variable "runtime" {
    description   = "Name of the runtime. Defaults to 'dotnetcore2.1'"
    type          = string
```

```
    default       = "dotnetcore2.1"
}

variable "memory_size" {
    description   = "Memory allocation for the Lambda in Megabytes"
    type          = number
    default       = 256
}

variable "timeout" {
    description   = "Timeout period in seconds"
    type          = number
}

variable "role_arn" {
    description   = "ARN of the role under which the Lambda should execute"
    type          = string
}

variable "publish" {
    description   = "Whether to publish creation/change as new Lambda Function
Version"
    type          = bool
    default       = false
}

variable "environment_variables" {
    description = "Environment variables for the lambda. Use to supply
configuration information"
    type        = map
    default     = {}
}

variable "lambda_alias_current" {
    description = "Name of the alias being created."
    type        = string
}

variable "subnet_ids" {
```

```
    description = "A list of subnet IDs associated with the Lambda Function"
    type        = list
    default     = []
}

variable "security_group_ids" {
    description = "A list of security group IDs associated with the Lambda
Function"
    type        = list
    default     = []
}
```

Файл lambda_role main.tf

```
data "aws_iam_policy_document" "lambda_can_assume_role" {
    statement {
      actions = [ "sts:AssumeRole" ]
      principals {
        type = "Service"
        identifiers = [ "lambda.amazonaws.com" ]
      }
    }
}

resource "aws_iam_role" "lambda_role" {
  name                  = var.iam_role_name
  assume_role_policy    =
data.aws_iam_policy_document.lambda_can_assume_role.json
  force_detach_policies = true
  tags                  = var.tags
}
```

Файл lambda_role output.tf

```
output "name" {
    description = "AWS IAM Role name"
    value = aws_iam_role.lambda_role.name
}

output "arn" {
    description = "AWS IAM Role arn"
    value = aws_iam_role.lambda_role.arn
```

```
}
```

Файл lambda_role variables.tf

```
variable "iam_role_name" {
    description = "IAM role name"
    type        = string
}

variable "tags" {
    description = "Addition tags for resources"
    type = map
    default = {}
}
```

Файл sns main.tf

```
resource aws_sns_topic "sns" {
    name = var.sns_name
}
```

Файл sns output.tf

```
output "arn" {
    description = "SNS topic ARN"
    value       = aws_sns_topic.sns.arn
}
```

Файл sns variables.tf

```
variable "sns_name" {
    description = "SNS Topic name"
    type        = string
}
```

Файл sqs main.tf

```
resource "aws_sqs_queue" "sqs" {
    name                        = var.sqs_name
    visibility_timeout_seconds = var.visibility_timeout_seconds
    redrive_policy              = var.redrive_policy
}
```

Файл sns output.tf

```
output "arn" {
    description = "SQS Amazon Resource Number"
    value       = aws_sqs_queue.sqs.arn
}
```

```
output "url" {
    description = "The URL for the created Amazon SQS queue"
    value       = aws_sqs_queue.sqs.id
}
```

Файл sns variables.tf

```
variable "sqs_name" {
    description = "SQS name"
    type        = string
}

variable "visibility_timeout_seconds" {
    description = "The visibility timeout for the queue. Default is 30."
    type        = number
    default     = 30
}

variable "redrive_policy" {
    description = "JSON policy to set up the Dead Letter Queue"
    type        = string
    default     = null
}
```

Файл _api-gateway.tf

```
module "api_gateway" {
    source      = "./modules/api_gateway"

    name         = "${var.prefix}-api-gateway"
    stage_name   = "dev"
    lambda_names = [
        module.user-management-lambda.lambda_name,
        module.license-management-lambda.lambda_name,
        module.product-management-lambda.lambda_name
    ]
    users_uri    = module.user-management-lambda.lambda_invoke_arn
    license_uri  = module.license-management-lambda.lambda_invoke_arn
    products_uri = module.product-management-lambda.lambda_invoke_arn
}
```

Файл _dynamodb.tf

```
module "dynamodb_users_table" {
    source = "./modules/dynamodb"

    name = "${var.prefix}-Users"
    billing_mode = var.billing_mode
    read_capacity = var.read_capacity
    write_capacity = var.write_capacity
    hash_key = "Uuid"

    attributes = [
        {
            name = "Uuid"
            type = "S"
        }
    ]
}

module "dynamodb_product_table" {
    source = "./modules/dynamodb"

    name = "${var.prefix}-Products"
    billing_mode = var.billing_mode
    read_capacity = var.read_capacity
    write_capacity = var.write_capacity
    hash_key = "ProductId"

    attributes = [
        {
            name = "ProductId"
            type = "S"
        }
    ]
}

module "dynamodb_license_table" {
    source = "./modules/dynamodb"
```

```
    name = "${var.prefix}-Licenses"
    billing_mode = var.billing_mode
    read_capacity = var.read_capacity
    write_capacity = var.write_capacity
    hash_key = "LicenseId"

    attributes = [
        {
            name = "LicenseId"
            type = "S"
        }
    ]
}

module "dynamodb_state_table" {
    source = "./modules/dynamodb"

    name = "${var.prefix}-CircuitBreakerState"
    billing_mode = var.billing_mode
    read_capacity = var.read_capacity
    write_capacity = var.write_capacity
    hash_key = var.hash_key

    attributes = var.attributes
}

module "dynamodb_entitlements_table" {
    source = "./modules/dynamodb"

    name = "${var.prefix}-Entitlements"
    billing_mode = var.billing_mode
    read_capacity = var.read_capacity
    write_capacity = var.write_capacity
    hash_key = "EntitlementId"

    attributes = [
        {
            name = "EntitlementId"
            type = "S"
```

```
        }
    ]
}
```

Файл _eventbridge.tf

```
resource "aws_cloudwatch_event_rule" "lambda_schedule" {
    name                = "${var.prefix}-schedule"
    is_enabled          = false
    schedule_expression = "rate(300 days)"
}

resource "aws_cloudwatch_event_target" "lambda_target" {
    rule      = aws_cloudwatch_event_rule.lambda_schedule.name
    target_id = aws_cloudwatch_event_rule.lambda_schedule.name
    arn       = module.user-integration-lambda.lambda_arn
    input     = jsonencode({
        "IsMaintenance" = true,
        "Action": "CircuitBreakerTrial"
    })
}

resource "aws_lambda_permission" "allow_cloudwatch_to_call_lambda" {
    statement_id  = "AllowExecutionFromCloudWatch"
    action        = "lambda:InvokeFunction"
    function_name = module.user-integration-lambda.lambda_name
    principal     = "events.amazonaws.com"
    source_arn    = aws_cloudwatch_event_rule.lambda_schedule.arn
}
```

Файл _license-management-lambda-addition.tf

```
resource "aws_iam_policy" "license_management_dynamoDB_policy" {
    name = "${var.prefix}-${var.license_management_lambda_name}-dynamodb-policy"

    policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "dynamodb:DescribeTable",
```

```
          "dynamodb:DeleteItem",
          "dynamodb:UpdateItem",
          "dynamodb:PutItem",
          "dynamodb:GetItem",
          "dynamodb:Scan"
        ]
        Resource = [ "${module.dynamodb_license_table.arn}",
"${module.dynamodb_entitlements_table.arn}" ]
      }
    ]
  })
}

resource "aws_iam_policy" "license_management_sqs_policy" {
  name = "${var.prefix}-${var.license_management_lambda_name}-sqs-policy"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "sqs:ReceiveMessage",
          "sqs:DeleteMessage",
          "sqs:GetQueueAttributes"
        ]
        Resource = [ "${module.license-management-lambda-sqs.arn}" ]
      }
    ]
  })
}

resource "aws_iam_role_policy_attachment" "license-management-dynamodb-access" {
    role       = module.license-management-lambda-role.name
    policy_arn = aws_iam_policy.license_management_dynamoDB_policy.arn
}

resource "aws_iam_role_policy_attachment" "license_management_sqs_access" {
    role       = module.license-management-lambda-role.name
```

```
    policy_arn = aws_iam_policy.license_management_sqs_policy.arn
}

resource "aws_iam_role_policy_attachment" "license_management_lambda_logs" {
  role         = module.license-management-lambda-role.name
  policy_arn   = "arn:aws:iam::aws:policy/service-
role/AWSLambdaBasicExecutionRole"
}
```

Файл _license-management-lambda.tf

```
module "license-management-lambda-role" {
    source          = "./modules/lambda-role"
    iam_role_name   = "${var.prefix}-role-${var.license_management_lambda_name}-
${var.aws_region}"
}

module "license-management-lambda" {
    source = "./modules/lambda"

    prefix               = var.prefix
    name                 = var.license_management_lambda_name
    role_arn             = module.license-management-lambda-role.arn
    description          = var.license_management_lambda_description
    filename             = var.license_management_lambda_filename
    runtime              = var.runtime
    handler              = var.license_management_lambda_handler
    memory_size          = var.memory_size
    timeout              = var.lambda_timeout
    publish              = var.publish
    lambda_alias_current = var.lambda_alias_current

    environment_variables = {
        "SWAGGER_ENABLED" = true
        "Parameters__ProductsApiUrl" =
"${module.api_gateway.invoke_url}${var.products-prefix}/"
        "Parameters__UsersApiUrl"    =
"${module.api_gateway.invoke_url}${var.users-prefix}/"
    }
}
```

```
# SQS Lambda Event Source
resource "aws_lambda_event_source_mapping" "license-management-sqs-event-source"
{
    event_source_arn = module.license-management-lambda-sqs.arn
    function_name    = module.license-management-lambda.lambda_arn
    batch_size       = 1
}
```

Файл _product-management-lambda-addition.tf

```
resource "aws_iam_policy" "product_management_dynamoDB_policy" {
    name = "${var.prefix}-${var.product_management_lambda_name}-dynamodb-policy"

    policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "dynamodb:DescribeTable",
          "dynamodb:DeleteItem",
          "dynamodb:UpdateItem",
          "dynamodb:PutItem",
          "dynamodb:GetItem",
          "dynamodb:Scan"
        ]
        Resource = [ "${module.dynamodb_product_table.arn}" ]
      }
    ]
  })
}

resource "aws_iam_policy" "product_management_snsPublish_policy" {
  name = "${var.prefix}-${var.product_management_lambda_name}-sns-policy"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
```

```
        Effect = "Allow"
        Action = [
          "sns:Publish"
        ]
        Resource = [ "${module.product-management-lambda-sns.arn}" ]
      }
    ]
  })
}

resource "aws_iam_role_policy_attachment" "product_management_dynamodb_access" {
    role       = module.product-management-lambda-role.name
    policy_arn = aws_iam_policy.product_management_dynamoDB_policy.arn
}

resource "aws_iam_role_policy_attachment" "product_management_sns_access" {
    role       = module.product-management-lambda-role.name
    policy_arn = aws_iam_policy.product_management_snsPublish_policy.arn
}

resource "aws_iam_role_policy_attachment" "product_management_lambda_logs" {
  role         = module.product-management-lambda-role.name
  policy_arn   = "arn:aws:iam::aws:policy/service-
role/AWSLambdaBasicExecutionRole"
}
```

Файл _product-management-lambda.tf

```
module "product-management-lambda-role" {
    source          = "./modules/lambda-role"
    iam_role_name   = "${var.prefix}-role-${var.product_management_lambda_name}-
${var.aws_region}"
}

module "product-management-lambda" {
    source = "./modules/lambda"

    prefix               = var.prefix
    name                 = var.product_management_lambda_name
    role_arn             = module.product-management-lambda-role.arn
```

```
    description          = var.product_management_lambda_description
    filename             = var.product_management_lambda_filename
    runtime              = var.runtime
    handler              = var.product_management_lambda_handler
    memory_size          = var.memory_size
    timeout              = var.lambda_timeout
    publish              = var.publish
    lambda_alias_current = var.lambda_alias_current

    environment_variables = {
        "SWAGGER_ENABLED" = true
        "Parameters__SnsTopicArn"    = module.product-management-lambda-sns.arn
    }
}
```

Файл _sns.tf

```
module "user-management-lambda-sns" {
    source = "./modules/sns"

    sns_name = "${var.prefix}-${var.user_management_lambda_name}-sns"
}

module "product-management-lambda-sns" {
    source = "./modules/sns"

    sns_name = "${var.prefix}-${var.product_management_lambda_name}-sns"
}

data "aws_iam_policy_document" "users_integration_put_to_sqs" {
  statement {
    sid    = "PutMessagesToSQS"
    effect = "Allow"

    principals {
      type        = "Service"
      identifiers = ["sns.amazonaws.com"]
    }

    actions   = ["sqs:SendMessage"]
```

```
    resources = [ module.user-integration-lambda-sqs.arn ]

    condition {
      test     = "ArnEquals"
      variable = "aws:SourceArn"
      values   = [ module.user-management-lambda-sns.arn]
    }
  }
}

resource "aws_sqs_queue_policy" "users_subscription" {
  queue_url = module.user-integration-lambda-sqs.url
  policy = data.aws_iam_policy_document.users_integration_put_to_sqs.json
}

data "aws_iam_policy_document" "license_management_put_to_sqs" {
  statement {
    sid    = "PutMessagesToSQS"
    effect = "Allow"

    principals {
      type        = "Service"
      identifiers = ["sns.amazonaws.com"]
    }

    actions   = ["sqs:SendMessage"]
    resources = [ module.license-management-lambda-sqs.arn ]

    condition {
      test     = "ArnEquals"
      variable = "aws:SourceArn"
      values   = [ module.user-management-lambda-sns.arn, module.product-
management-lambda-sns.arn]
    }
  }
}

resource "aws_sqs_queue_policy" "license_subscription" {
  queue_url = module.license-management-lambda-sqs.url
```

```
  policy = data.aws_iam_policy_document.license_management_put_to_sqs.json
}
```

Файл _sqs.tf

```
module "user-integration-lambda-sqs" {
    source         = "./modules/sqs"

    sqs_name       = "${var.prefix}-${var.user_integration_lambda_name}-sqs"
    redrive_policy = "{\"deadLetterTargetArn\" : \"${module.user-integration-
lambda-dlq.arn}\", \"maxReceiveCount\": 4}"
}

module "user-integration-lambda-dlq" {
    source = "./modules/sqs"

    sqs_name = "${var.prefix}-${var.user_integration_lambda_name}-
deadletterqueue"
}

resource "aws_sns_topic_subscription" "user_integration_sqs_target" {
    topic_arn            = module.user-management-lambda-sns.arn
    protocol             = "sqs"
    endpoint             = module.user-integration-lambda-sqs.arn
    raw_message_delivery = true
}

module "license-management-lambda-sqs" {
    source         = "./modules/sqs"

    sqs_name       = "${var.prefix}-${var.license_management_lambda_name}-sqs"
    redrive_policy = "{\"deadLetterTargetArn\" : \"${module.license-management-
lambda-dlq.arn}\", \"maxReceiveCount\": 4}"
}

module "license-management-lambda-dlq" {
    source = "./modules/sqs"

    sqs_name = "${var.prefix}-${var.license_management_lambda_name}-
deadletterqueue"
```

```
}

resource "aws_sns_topic_subscription" "license_management_sqs_users_target" {
    topic_arn            = module.user-management-lambda-sns.arn
    protocol             = "sqs"
    endpoint             = module.license-management-lambda-sqs.arn
    raw_message_delivery = true
    filter_policy         = "${jsonencode(tomap({"Action" = tolist(["Delete",
"Update"])}))}"
}

resource "aws_sns_topic_subscription" "license_management_sqs_products_target" {
    topic_arn            = module.product-management-lambda-sns.arn
    protocol             = "sqs"
    endpoint             = module.license-management-lambda-sqs.arn
    raw_message_delivery = true
    filter_policy         = "${jsonencode(tomap({"Action" = tolist(["Delete",
"Update"])}))}"
}
```

Файл _user-integration-lambda-addition.tf

```
resource "aws_iam_policy" "user_integration_dynamoDB_policy" {
    name = "${var.prefix}-${var.user_integration_lambda_name}-dynamodb-policy"

    policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "dynamodb:DescribeTable",
          "dynamodb:DeleteItem",
          "dynamodb:UpdateItem",
          "dynamodb:PutItem",
          "dynamodb:GetItem",
          "dynamodb:Scan"
        ]
        Resource = [ "${module.dynamodb_state_table.arn}",
"${module.dynamodb_users_table.arn}" ]
```

```
      }
    ]
  })
}

resource "aws_iam_policy" "user_integration_sqs_policy" {
  name = "${var.prefix}-${var.user_integration_lambda_name}-sqs-policy"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "sqs:ReceiveMessage",
          "sqs:DeleteMessage",
          "sqs:GetQueueAttributes"
        ]
        Resource = [ "${module.user-integration-lambda-sqs.arn}" ]
      },
      {
        Effect = "Allow"
        Action = [
          "sqs:SendMessage"
        ]
        Resource = [ "${module.user-integration-lambda-dlq.arn}" ]
      }
    ]
  })
}

resource "aws_iam_policy" "user_integration_event_mapping_policy" {
  name = "${var.prefix}-${var.user_integration_lambda_name}-event-mapping-policy"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
```

```
        Action = [
          "lambda:ListEventSourceMappings",
          "lambda:UpdateEventSourceMapping"
        ]
        Resource = [ "*" ]
      },
      {
        Effect = "Allow"
        Action = [
          "events:PutRule"
        ]
        Resource = [ aws_cloudwatch_event_rule.lambda_schedule.arn ]
      }
    ]
  })
}

resource "aws_iam_role_policy_attachment" "user_integration_dynamodb_access" {
    role       = module.user-integration-lambda-role.name
    policy_arn = aws_iam_policy.user_integration_dynamoDB_policy.arn
}

resource "aws_iam_role_policy_attachment" "user_integration_sqs_access" {
    role       = module.user-integration-lambda-role.name
    policy_arn = aws_iam_policy.user_integration_sqs_policy.arn
}

resource "aws_iam_role_policy_attachment" "user_integration_lambda_logs" {
  role         = module.user-integration-lambda-role.name
  policy_arn   = "arn:aws:iam::aws:policy/service-
role/AWSLambdaBasicExecutionRole"
}

resource "aws_iam_role_policy_attachment" "user_integration_event_mapping" {
  role         = module.user-integration-lambda-role.name
  policy_arn   = aws_iam_policy.user_integration_event_mapping_policy.arn
}
```

Файл _user-integration-lambda.tf

```
module "user-integration-lambda-role" {
    source          = "./modules/lambda-role"
    iam_role_name   = "${var.prefix}-role-${var.user_integration_lambda_name}-
${var.aws_region}"
}

module "user-integration-lambda" {
    source = "./modules/lambda"

    prefix               = var.prefix
    name                 = var.user_integration_lambda_name
    role_arn             = module.user-integration-lambda-role.arn
    description          = var.user_integration_lambda_description
    filename             = var.user_integration_lambda_filename
    runtime              = var.runtime
    handler              = var.user_integration_lambda_handler
    memory_size          = var.memory_size
    timeout              = var.lambda_timeout
    publish              = var.publish
    lambda_alias_current = var.lambda_alias_current

    environment_variables = {
        "SWAGGER_ENABLED"                       = true
        "Serilog__MinimumLogLevel"              = "Debug"
        "Parameters__UsersTableName"            =
module.dynamodb_users_table.name
        "EventBridge__RuleName"                 =
aws_cloudwatch_event_rule.lambda_schedule.id
        "CircuitBreaker__CircuitStateTableName" =
module.dynamodb_state_table.name
        "CircuitBreaker__ErrorThreshold"        = 1
        "CircuitBreaker__SamplingDuration"      = "00:05:00"
        "CircuitBreaker__Timeout"               = "00:01:00"
        "CircuitBreaker__DeadLetterQueueUrl"    = module.user-integration-lambda-
dlq.url
        "CircuitBreaker__SourceQueueUrl"        = module.user-integration-lambda-
sqs.url
    }
}
```

```
# SQS Lambda Event Source
resource "aws_lambda_event_source_mapping" "users-integration-sqs-event-source" {
    event_source_arn = module.user-integration-lambda-sqs.arn
    function_name    = module.user-integration-lambda.lambda_arn
    batch_size       = 1
}
```

Файл _user-management-lambda-addition.tf

```
resource "aws_iam_policy" "user_management_dynamoDB_policy" {
    name = "${var.prefix}-${var.user_management_lambda_name}-dynamodb-policy"

    policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "dynamodb:DescribeTable",
          "dynamodb:DeleteItem",
          "dynamodb:UpdateItem",
          "dynamodb:PutItem",
          "dynamodb:GetItem",
          "dynamodb:Scan"
        ]
        Resource = [ "${module.dynamodb_users_table.arn}" ]
      }
    ]
  })
}

resource "aws_iam_policy" "user_management_snsPublish_policy" {
  name = "${var.prefix}-${var.user_management_lambda_name}-sns-policy"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
```

```
        Action = [
          "sns:Publish"
        ]
        Resource = [ "${module.user-management-lambda-sns.arn}" ]
      }
    ]
  })
}

resource "aws_iam_role_policy_attachment" "user-management-dynamodb-access" {
    role       = module.user-management-lambda-role.name
    policy_arn = aws_iam_policy.user_management_dynamoDB_policy.arn
}

resource "aws_iam_role_policy_attachment" "user-management-sns-access" {
    role       = module.user-management-lambda-role.name
    policy_arn = aws_iam_policy.user_management_snsPublish_policy.arn
}

resource "aws_iam_role_policy_attachment" "user_management_lambda_logs" {
  role         = module.user-management-lambda-role.name
  policy_arn   = "arn:aws:iam::aws:policy/service-
role/AWSLambdaBasicExecutionRole"
}
```

Файл _ user-management-lambda.tf

```
module "user-management-lambda-role" {
    source          = "./modules/lambda-role"
    iam_role_name   = "${var.prefix}-role-${var.user_management_lambda_name}-
${var.aws_region}"
}

module "user-management-lambda" {
    source = "./modules/lambda"

    prefix               = var.prefix
    name                 = var.user_management_lambda_name
    role_arn             = module.user-management-lambda-role.arn
    description          = var.user_management_lambda_description
```

```
    filename             = var.user_management_lambda_filename
    runtime              = var.runtime
    handler              = var.user_management_lambda_handler
    memory_size          = var.memory_size
    timeout              = var.lambda_timeout
    publish              = var.publish
    lambda_alias_current = var.lambda_alias_current

    environment_variables = {
        "SWAGGER_ENABLED"            = true
        "Serilog__MinimumLogLevel"   = "Debug"
        "Parameters__SnsTopicArn"    = module.user-management-lambda-sns.arn
    }
}
```

Файл infra.auto.tfvars

```
# Providers
aws_region = "eu-central-1"

# Shared
prefix = "LicenseManagement"

# Lambda Common
runtime                      = "dotnet6"
memory_size                  = "512"
lambda_timeout               = "15"
publish                      = true
lambda_alias_current         = "current_version"

# Users Management Lambda
user_management_lambda_name        = "users-management-lambda"
user_management_lambda_description = "Automated deployment of Users Management
Lambda"
user_management_lambda_handler     =
"UserManagementLambda::UserManagementLambda.LambdaEntryPoint::FunctionHandlerAsyn
c"

# Users Integration Lambda
user_integration_lambda_name        = "users-integration-lambda"
```

```
user_integration_lambda_description = "Automated deployment of Users Integration
Lambda"
user_integration_lambda_handler     =
"UserIntegrationLambda::UserIntegrationLambda.Function::FunctionHandler"

# License Management Lambda
license_management_lambda_name        = "license-management-lambda"
license_management_lambda_description = "Automated deployment of License
Management Lambda"
license_management_lambda_handler     =
"LicenseManagementLambda::LicenseManagementLambda.Function::FunctionHandlerAsync"

# Product Management Lambda
product_management_lambda_name        = "product-management-lambda"
product_management_lambda_description = "Automated deployment of Product
Management Lambda"
product_management_lambda_handler     =
"ProductManagementLambda::ProductManagementLambda.LambdaEntryPoint::FunctionHandl
erAsync"

# Dynamo DB
billing_mode    = "PAY_PER_REQUEST"
read_capacity   = "0"
write_capacity  = "0"
hash_key        = "Id"

attributes      = [
    {
        name    = "Id"
        type    = "S"
    }
]
```

Файл main.tf

```
terraform {
    required_providers {
      aws = {
        source = "hashicorp/aws"
      }
```

```
    }

    backend "s3" {
      bucket  = "licensemanagement-tfstate"
      key     = "state/terraform.tfstate"
      region  = "eu-central-1"
      encrypt = true
  }
}

provider "aws" {
    region =  var.aws_region
    shared_credentials_files = [ "~/.aws/credentials" ]
}
```

Файл variables.tf

```
# Provider
variable "aws_region" {
  type = string
}

# Shared
variable "prefix" {
  type = string
}

variable "products-prefix" {
  type = string
  default = "products-api"
}

variable "users-prefix" {
  type = string
  default = "users-api"
}

variable "licenses-prefix" {
  type = string
  default = "license-api"
```

```
}

# User Management Lambda
variable "user_management_lambda_name" {
  type = string
}
variable "user_management_lambda_description" {
  type = string
}
variable "user_management_lambda_handler" {
  type = string
}
variable "user_management_lambda_filename" {
  type = string
}

#User Integration Lambda
variable "user_integration_lambda_name" {
  type = string
}
variable "user_integration_lambda_description" {
  type = string
}
variable "user_integration_lambda_handler" {
  type = string
}
variable "user_integration_lambda_filename" {
  type = string
}

#License Management Lambda
variable "license_management_lambda_name" {
  type = string
}
variable "license_management_lambda_description" {
  type = string
}
variable "license_management_lambda_handler" {
  type = string
```

```
}
variable "license_management_lambda_filename" {
  type = string
}

#Product Management Lambda
variable "product_management_lambda_name" {
  type = string
}
variable "product_management_lambda_description" {
  type = string
}
variable "product_management_lambda_handler" {
  type = string
}
variable "product_management_lambda_filename" {
  type = string
}

# Lambda Common
variable "runtime" {
  type = string
  default = "dotnet6"
}
variable "memory_size" {
  type = string
  default = "512"
}
variable "lambda_timeout" {
  type = string
  default = "15"
}
variable "publish" {
  type = bool
  default = true
}
variable "lambda_alias_current" {
  type = string
  default = "current_version"
```

```
}

# DynamoDB
 variable "billing_mode" {
  type = string
 }
 variable "read_capacity" {
  type = string
 }
 variable "write_capacity" {
  type = string
 }
 variable "hash_key" {
  type = string
 }
 variable "attributes" {
  type = any
 }
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ХМАРНЕ АРХІТЕКТУРНЕ РІШЕННЯ СЕРВІСУ ОБРОБКИ ЛІЦЕНЗІЙ ПРОДУКТІВ ТА ПОСЛУГ

Програма та методика тестування

КПІ.ІТ-84в23.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Мирослав ОСАДЧИЙ

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є інтегрований модуль з обробки ліцензій продуктів та послуг. Розроблено з використанням середовища .NET Core 6 та AWS. Протестований у веб-браузерах Google Chrome та Microsoft Edge з операційною системою Windows 11.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка збереження даних;

– перевірка сумісності веб-додатку з останніми версіями сучасних браузерів (Chrome, Opera, Firefox, Microsoft Edge);

– перевірка сумісності застосунку з різними операційними системами;

– знаходження проблем, помилок і недоліків з метою їх усунення;

– перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;

– динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;

– системне тестування – перевіряється усе програмне забезпечення в цілому;

– мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;

– тестування «чорної скриньки» – об’єктом тестування тут є функції присутні у програмі. Перевіряється коректність вихідних даних при заданих вхідних;

– тестування «білої скриньки» – об’єктом тестування тут є внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним;

– тестування «сірої скриньки» – об’єктом тестування тут є деякі особливості внутрішньої поведінки програми. Перевіряється коректність вихідних даних при заданих вхідних, застосовується для тестування окремих алгоритмів (функцій).

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з використанням наскрізного тестування (End-to-end, E2E), з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

– динамічне тестування на відповідність функціональним вимогам;

– тестування на виведення повідомлень про помилку, коли це необхідно;

– тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ХМАРНЕ АРХІТЕКТУРНЕ РІШЕННЯ СЕРВІСУ ОБРОБКИ ЛІЦЕНЗІЙ ПРОДУКТІВ ТА ПОСЛУГ

Керівництво користувача

КПІ.ІТ-84в23.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Катерина ЛІЩУК

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Мирослав ОСАДЧИЙ

Київ – 2023

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Хмарне архітектурне рішення сервісу обробки ліцензій продуктів та послуг – це інтегрований програмний модуль, створений для уніфікації обробки доступів у бізнес застосунку. Реалізація легко розгортується, масштабується, взаємо заміняється та інтегрується у сучасне програмне забезпечення, потребує мінімальних зусиль у підтриманні.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних мінімальних вимог:

– ПК зі встановленим браузером Google Chrome, Microsoft Edge, Mozilla Firefox, Opera останніх версій;

– об’єм ОЗП: 1 Гб;

– наявність доступу до Інтернету;

– процесор сімейства Intel Core i3.

Для оптимальної роботи даного застосунку також необхідне виконання наступних рекомендованих вимог:

– об‘єм ОЗП: 16 Гб;

– підключення до мережі Інтернет зі швидкістю від 100 мегабіт.

2.2 Завантаження застосунку

Застосунок можна розгорнути за допомогою Jenkins пайплайну, код якого приведений у тексті програми. Для цього необхідно створити Jenkins аккаунт, надати йому AWS обліковий запис та запустити. Усі необхідні файли та інфраструктура буде створені автоматично.

Також, інфраструктуру модуля можна розгорнути вручну: для цього необхідно завантажити тераформ скрипти з репозиторіюі і виконати команди terraform init та terraform apply у їх головній директорії (де знаходиться файл main.tf).

2.3 Перевірка коректної роботи

Для перевірити коректної роботи застосунку, можна перейти за адресою, яку створить розгорнутий API Gateway, та переконатись у доступності ендпоінтів {url}/users-api {url}/license-api {url}/products-api.

3 ВИКОНАННЯ ПРОГРАМИ

Дане програмне забезпечення не є повноцінним додатком, а лише архітектурним рішенням з програмним інтерфейсом. Воно є інтегрованим open-source модулем, тому інструкції по його роботі можуть значно відрізнятися при використанні у різних системах та бізнес застосунку.