НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

«На правах рукопису»
УДК _004.67

До захисту допущено:
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____2022 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою «Інтегровані інформаційні системи»
зі спеціальності 126 «Інформаційні системи та технології»
на тему: «Фреймворк для створення веб-сервісів управління ресурсами підприємства»

Виконав:
студент VI курсу, групи ІА-12мп
Тарасенко Артем Дмитрович _____

Керівник:
Професор каф. ІСТ, д.ф.-м.н., професор
Дорошенко Анатолій Юхимович _____

Рецензент:
Професор каф. ІПІ, д.т.н., професор
Стеценко Інна В’ячеславівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2022 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – другий (магістерський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інтегровані інформаційні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____2022 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Тарасенко Артему Дмитровичу

1. Тема дисертації «Фреймворк для створення веб-сервісів управління ресурсами підприємств», науковий керівник дисертації Дорошенко Анатолій Юхимович професор кафедри ІСТ д.ф.-м.н. затверджені наказом по університету від «09» 11 2022 р. № 4115-с

2. Термін подання студентом дисертації 12.12.2022

3. Об’єкт дослідження: фреймворк для запуску веб-сервісів управління ресурсами підприємства.

4. Вихідні дані: програмна записка

5. Перелік завдань, які потрібно розробити: створити фреймворк для запуску веб-сервісів систем управління ресурсами підприємства, провести його тестування, оцінити можливість його реалізації в якості комерційного сервісу.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:

1. Загальна структурна схема фреймворку
2. Блок-схема алгоритму підпису запиту
3. Схема маршрутизації запитів у API шлюзі
4. Схема взаємодії хмарних сервісів AWS

5. ER діаграма інфраструктури фреймворку

6. Діаграма прецедентів

7. Діаграма послідовності запуску ERP системи

8. Діаграма послідовності видалення ERP системи

7. Орієнтовний перелік публікацій А.Д. Тарасенко, А.Ю. Дорошенко. Веб-сервіс для систем управління ресурсами підприємства. Проблеми програмування, 2022. № 2. С. 47–56.

8. Дата видачі завдання 05.09.2022

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Затвердження теми роботи	5.09.2022 – 09.09.2022	Виконано
2	Визначення та аналіз предметної області	12.09.2022 – 16.09.2022	Виконано
3	Аналіз існуючих аналогів системи	19.09.2022 – 30.09.2022	Виконано
4	Вибір технологій та початок розробки	03.10.2022 – 28.10.2022	Виконано
5	Оформлення дисертації	31.10.2022 – 05.12.2022	Виконано

Студент Артем ТАРАСЕНКО

Науковий керівник Анатолій ДОРОШЕНКО

АНОТАЦІЯ

Тарасенко А.Д. Фреймворк для створення веб-сервісів управління ресурсами підприємства. КПІ ім. Ігоря Сікорського, Київ, 2022

Дисертація складається з 127 сторінок, 69 рисунків, 23 таблиць, 27 літературних джерел та 8 додатків.

Метою даної магістерської дисертації є реалізація фреймворку для запуску веб-сервісів ERP систем зі здатністю до розширення шляхом інсталяції ERP модулів. Завдяки даному фреймворку користувачі зможуть швидко запускати ERP системи із потрібними налаштуваннями в залежності від потреб їхнього бізнесу.

Актуальність розробки зумовлена постійним швидкістю розвиту сфери IT та розширення впровадження систем управління ресурсами підприємства в усіх видах бізнесу – від малого до великого. Початок будь-якого бізнесу починається саме із розробки і впровадження даної системи під ті чи інші його потреби. Тому можливість налаштування даної системи за пару хвилин буде до вподоби усім.

Об’єктом дослідження є фреймворк для запуску веб-сервісів управління ресурсами підприємства.

Предметом дослідження є покращення і полегшення процесу впровадження систем управління ресурсами підприємства.

В ході виконання магістерської роботи було розроблено інформаційну систему взаємопов’язаних застосунків із використанням хмарних сервісів та технологій для управління фреймворком для створення систем управління ресурсами підприємства на основі необхідних користувачеві модулів. Проведено детальний аналіз існуючих аналогів ERP систем для формування загальної концепції та ідеї та реалізації оптимальної структури системи. Значну увагу було приділено безпеці та надійності інформації, а також можливості інтеграції системи із іншими додатками.

Ключові слова: ресурси підприємства, фреймворк, веб-сервіс, система управління, ERP система, веб-застосунок, адміністрування, хмарні обчислення.

SUMMARY

Tarasenko A.D. Framework for creating managing business resources web services. Igor Sikorsky KPI, Kyiv, 2022.

The Master`s Thesis consists of 127 pages, 69 figures, 23 tables, 27 literary sources and 8 appendixes.

The goal of this master's thesis is a framework for launching web services of the ERP system with the ability to expand by installing ERP modules. Thanks to this framework, users can quickly launch ERP systems with the necessary settings depending on business needs. The relevance of the development is determined by the constant rapid development of the IT sphere and the expansion of the implementation of the enterprise resource management system in all types of business – from small to large. Any business begins precisely with the development and implementation of this system for certain needs. Therefore, the ability to configure this system in a couple of minutes will be to everyone's liking.

The object of the study is to improve and facilitate the process of implementing the enterprise resource management system.

The subject of the study is a framework for launching enterprise resource management web services.

In the course of the master's thesis, an information system of interconnected applications was developed using cloud services and technologies for managing the framework for creating an enterprise resource management system based on several user modules. A detailed analysis of existing analogues of the ERP system was carried out for the formation of a general concept and idea and implementation of the optimal structure of the system. Considerable attention was paid to the security and reliability of information, as well as the possibility of integrating the system with other applications.

Keywords: enterprise resources, framework, web service, management system, ERP system, web applications, administration, cloud computing.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ERP – Enterprise Resource Planning;

MRP – Manufacturing Resource Planning;

CRM – Customer Relationship Management;

БД – База Даних;

СУБД – Система Управління Базами Даних;

HTTP – Hyper Text Transfer Protocol;

HTTPS – Hyper Text Transfer Protocol Secure;

ESB – Enterprise Service Bus;

IAM – Identity and Access Management;

SSO – Single Sign-On;

MFA – Multi-factor Authentication;

BO – Back Office;

API – Application Programming Interface;

JSON – JavaScript Object Notation;

XML – eXtensible Markup Language;

LINQ – Language Integrated Query;

DI – Dependency Injection;

ASP – Active Server Pages;

DOM – Document Object Model;

SPA – Single Page Application;

UI – User Interface;

HTML – Hyper Text Markup Language;

JS – JavaScript;

PWA – Progressive Web Application;

RPC – Remote Procedure Call;

EF – Entity Framework;

ORM – Object-relational Mapper;

DDL – Data Definition Language;

DML – Data Manipulation Language;

CQRS – Command Query Responsibility Segregation;

CRUD – Create Read Update Delete;

DTO – Data Transfer Object;

RDBMS – Relational Database Management System;

ORDBMS – Object-Relational Database Management System;

SQL – Structured Query Language;

SDK – Software Development Kit;

AWS – Amazon Web Services;

GCP – Google Cloud Platform;

SaaS – Software as a Service;

IaaS – Infrastructure as a Service;

PaaS – Platform as a Service;

MS – Microsoft;

ООП – Об’єктно-Орієнтоване Програмування;

ERN – Entity Resource Name;

ARN – Amazon Resource Name;

GUID – Globally Unique Identifier;

HMAC – Hash-based Message Authentication Code;

SHA – Secure Hash Algorithm;

URI – Uniform Resource Identifier;

S3 – Simple Storage Service;

EC2 – Elastic Cloud Computing;

AMI – Amazon Machine Image;

EBS – Elastic Block Store;

VPC – Virtual Private Cloud;

RDS – Relational Database Service;

ELB – Elastic Load Balancing;

SES – Simple Email Service.

ВСТУП

Актуальність проблеми

Стандартний процес організації бізнесу будь-якого рівня, від малого до великого, починається із етапу встановлення системи управління та автоматизації бізнес-процесів, менеджменту її ресурсів із спільною базою даних тощо. У більшості випадків для вирішення цього питання бізнесу доводиться звертатися до послуг IT-спеціалістів для розробки даної системи. Такий процес відбувався впродовж десятиліть, що породило цілу низку схожих за функціоналом систем та застосунків, оскільки в рамках різних сфер бізнесу не існує монополії тієї або іншої компанії, звідси маємо багато компаній-конкурентів, які мали схожі системи. Тому із часом у деяких IT-компаній виникла ідея створення загальної системи із усіма основними функціями, та постачати її в якості IT-послуги. Єдина проблема була в тому, що кінцевий, повністю готовий необхідний функціонал ніколи достеменно не відомий, оскільки у кожного замовника або компанії обов’язково виникнуть свої специфічні потреби. Тому було вирішено створити свого роду «каркас» системи із базовим функціоналом, який, за потреби, може бути розширена шляхом встановлення додаткових сервісів та функціоналу, які, в свою чергу, вже розробляються конкретно за замовленням клієнта, і постачати дану систему в якості IT-послуги. В результаті клієнт має вигоду в тому, що йому більше не потрібно звертатися до IT-спеціалістів за розробкою системи, що економить йому час та фінанси, а постачальник даної системи має прибуток від клієнта за її користування.

Об’єкт дослідження

Об’єктом дослідження магістерської дисертації є фреймворк для створення веб-сервісів систем управління ресурсами підприємства систем в якості потенційної комерційної IT-послуги.

Предмет

Предметом дослідження магістерської дисертації є покращення і полегшення процесу впровадження системи автоматизації і управління ресурсами підприємства.

Мета і завдання

Метою і завданням магістерської дисертації є реалізація фреймворку для створення веб-сервісів ERP систем, використовуючи який потенційний клієнт матиме змогу запускати ERP системи із бажаними конфігураціями у хмарі із глобальним інтернет доменом та керувати ними.

Практичне значення

Система, що буде розроблена у даній роботі, призначена для використання в якості IT-послуги на потенційних комерційних основах. Кінцевими користувачами даної системи є адміністратори, IT-спеціалісти компаній, яким поставлена задача впровадження ERP системи у даній компанії, та звичайні працівники компаній.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

Система планування виробничих ресурсів (MRP II) — це інтегрована інформаційна система, яка використовується підприємствами. Планування виробничих ресурсів (MRP II) розвинулося на основі попередніх систем планування потреб у матеріалах (MRP) із додатковою можливістю інтеграції додаткових даних, таких як вимоги до робочої сили та фінансові потреби [1].

Система розроблена для централізації, інтеграції та обробки інформації для прийняття ефективних рішень у проєктуванні, плануванні, управлінні запасами та контролю витрат у виробництві.

І MRP, і MRP II вважаються попередниками системи планування ресурсів підприємства (ERP), процесу, за допомогою якого компанія (зазвичай виробник) керує та інтегрує важливі частини свого бізнесу.

Інформаційна система управління ERP об’єднує планування, закупівлі, інвентаризацію, продажі, маркетинг, фінанси, людські ресурси та інші сфери. ERP найчастіше використовується в програмному середовищі, оскільки було розроблено багато великих програм для допомоги компаніям в процесі впровадження ERP.

Розуміння планування виробничих ресурсів (MRP II)

MRP II — це комп’ютеризована система, яка використовує дані в реальному часі для створення детальних виробничих планів, які координують надходження комплектуючих матеріалів із наявністю обладнання та робочої сили. MRP II широко використовується сам по собі, але також використовується як модуль у більш просунутих системах планування ресурсів підприємства (ERP).

MRP II є розширенням оригінальної системи планування матеріальних потреб (MRP I). Планування потреб у матеріалах (MRP) було однією з перших інтегрованих інформаційних систем на основі програмного забезпечення, призначених для підвищення продуктивності бізнесу.

Інформаційна система планування потреб у матеріалах — це система, заснована

на прогнозі продажів, яка використовується для планування постачання сировини та її кількості, враховуючи припущення щодо техніки і робочих одиниць, необхідних для прогнозування продажів.

У 1980-х роках виробники зрозуміли, що їм потрібне програмне забезпечення, яке могло б підключитися до їхніх систем бухгалтерського обліку та передбачити їхні потреби в запасах. MRP II було надано як рішення, яке включало ці функції на додаток до всього, що надавав MRP I.

Реальні приклади програмного забезпечення MRP II

Нижче наведено невелику вибірку деяких популярних постачальників програмного забезпечення MRP II станом на початок 2020 року:

– IQMS;
– Fishbowl;
– FactoryEdge;
– Prodsmart;
– Oracle Netsuite Manufacturing Edition;
– Epicor;
– S2K Enterprise.

Порівняння MRP I та MRP II

Для всіх намірів і цілей MRP II фактично замінив програмне забезпечення MRP I. Більшість систем MRP II забезпечують усі функції системи MRP. Але окрім планування основного виробництва, опису матеріалів (BOM) і відстеження запасів, MRP II також додатково забезпечує функціональність у сфері логістики, маркетингу та загальних фінансів.

Наприклад, MRP II може враховувати змінні, яких MRP не враховує, такі як можливості обладнання та персоналу, що дає більш реалістичне та цілісне уявлення про операційні можливості компанії. Багато рішень MRP II також пропонують

можливості моделювання, які дозволяють оператору вводити змінні та спостерігати за ефектами. MRP II іноді називають системою замкнутого циклу, оскільки вона може надавати зворотний зв’язок щодо певних операцій. MRP I містив такі три основні функції:

– головне планування виробництва;

– опис матеріалів;

– відстеження запасів.

MRP II включає ці три, а також деякі додаткові:

– планування потужності техніки;

– прогнозування попиту;

– гарантія якості;

– загальний облік.

Система MRP II і сьогодні широко використовується виробничими компаніями як окреме рішення або як частина системи планування ресурсів підприємства (ERP). Програмна система Enterprise Resource Planning (ERP) вважається наступником програмного забезпечення MRP II.

Пакети ERP містять програми, що виходять далеко за межі виробництва. Це включає в себе все: від управління людськими ресурсами та відносинами з клієнтами до управління корпоративними активами..

Висновок до розділу 1

У цьому розділі ERP-системи були розглянуті предметною областю даного проєкту. Дано визначення MRP-системи як основного модуля ERP-системи та її призначення. Були також перераховані основні характеристики, якими повинна володіти система ERP. Це також важливо на етапі проєктування та розробки системи.

2 АНАЛІЗ ГОТОВИХ РІШЕНЬ

Перед початком проєктування та розробки фреймворку потрібно провести аналіз на основі існуючих аналогів ERP систем, визначити основні компоненти фреймворку, принципи їхньої роботи та яким чином вони взаємодіють між собою.

2.1 Модульність

Ключовим аспектом будь-якої ERP-системи є модульність. Основою цієї концепції є поділ загальних сутностей або функціональних можливостей на окремі блоки. У той же час модульність — це тип властивості системи, яка вимірює ступінь, наскільки сильно пов’язані компоненти в системі можуть бути розділені на окремі сутності або кластери, які взаємодіють один з одним більше, ніж інші. У сильно взаємопов’язаних системах із низьким рівнем модульності несправності в одному компоненті можуть каскадно поширюватися на інший, збільшуючи ризик відмови всієї системи. І навпаки, у системі з високим ступенем модульності збурення одного компонента краще стримуються, і менш імовірно, що вони поширюватимуться на інші компоненти.. Ідея модульності широко практикується в управлінні різними системами.

Система вважається «модульною», якщо, наприклад, її можна розділити на кілька компонентів, які можна комбінувати та поєднувати в різних конфігураціях. Компоненти можуть певним чином з’єднуватися, взаємодіяти або спільно використовувати ресурси (наприклад, дані), дотримуючись стандартизованих інтерфейсів. Модульний продукт — це система «слабко пов’язаних» компонентів, на відміну від тісно інтегрованого продукту, де кожен компонент спеціально розроблений для роботи з певними іншими компонентами в тісно пов’язаній системі.

У рамках ERP-системи модулі вважаються програмними пакетами зі своїми моделями, послугами та функціями [2]. Приклад змісту модуля зображено на рисунку 2.1

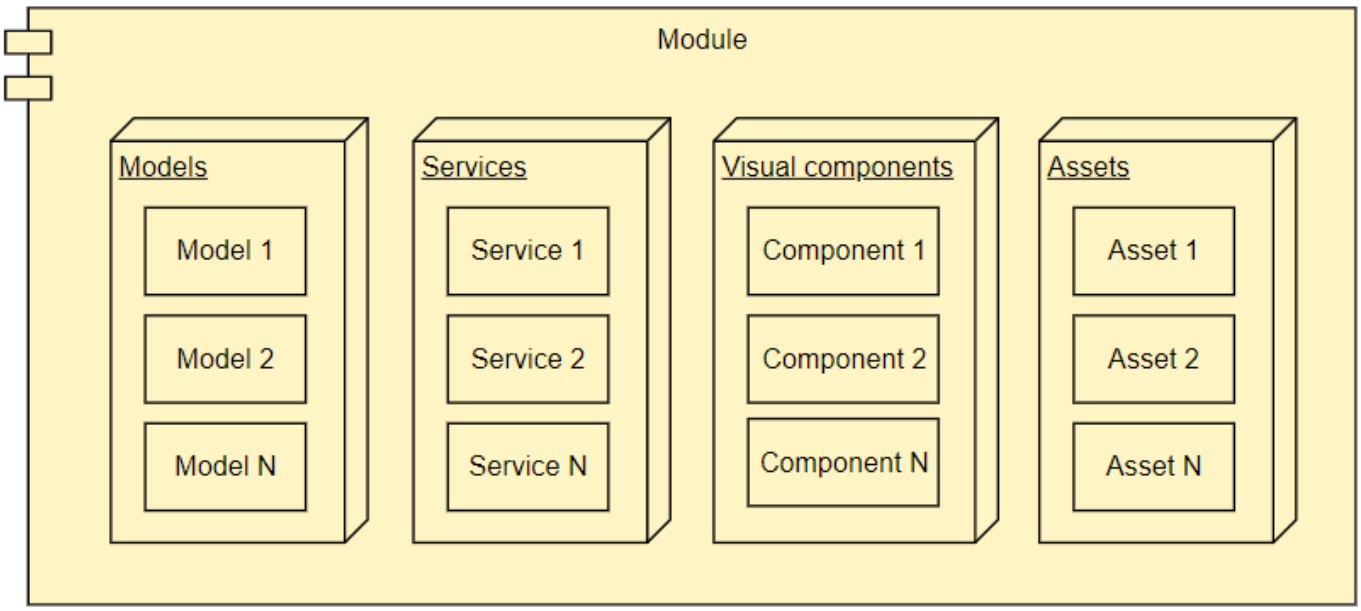

Рисунок 2.1 – Схема змісту програмного пакета модуля

Кожен модуль може бути реалізований як за вимогами клієнта, так і компанією-власником цього фреймворку. Клієнтам надається можливість вибирати модулі, необхідні їм для ведення свого бізнесу, тому власники фреймворку повинні розробити якнайбільше різних модулів, щоб залучити якнайбільше користувачів, що буде в їхніх інтересах.

2.2 Аналіз основних елементів

Як уже згадувалося в главі 1, ERP-системи призначені для централізації, консолідації та обробки інформації для ефективного прийняття рішень у галузі планування, проєктування, управління запасами та контролю витрат у виробничій інформаційній системі. Звідси можна виділити основні програмні елементи, які необхідні коректної роботи всієї системи:

— компонент роботи із базою даних (для зберігання корпоративної інформації компанії);

— компонент зв’язку і комунікації (для обробки зовнішніх запитів);

— головне програмне ядро (для управління взаємодією інших компонентів).

2.3 Функціонал за замовчуванням

Окрім функціоналу, впровадженого шляхом встановлення того або іншого модуля, у ERP системі має бути встановлений певний набір функціональних особливостей.

2.3.1 ERP безпека

Безпека ERP – це практика вжиття ефективних заходів безпеки для запобігання вторгненню у систему ERP. Захищена система ERP охоплює безпечну конфігурацію сервера та дозволяє вести журнал безпеки, забезпечувати безпеку зв'язку системи та безпеку даних. Користувачі та авторизації однаково важливі.

Безпека ERP часто є сліпою плямою для багатьох компаній. У міру того, як компанії витрачають час і ресурси на модернізацію своїх ERP-систем та елементів управління, вони часто не беруть до уваги або недостатньо інвестують у належну кібербезпеку. Атаки на ERP-системи можуть мати руйнівні наслідки діяльності компанії, оскільки вони беруть участь у багатьох повсякденних операціях [3]. І зі зростанням загроз кібербезпеці потенціал фінансових втрат і репутаційного збитку стає реальним. Організації повинні захищати свої системи від внутрішніх та зовнішніх кіберзагроз, щоб підтримувати конфіденційність, доступність та цілісність.

Належні засоби контролю за допомогою впровадження рішень безпеки ERP та їх інтеграції з іншими безпековими операціями мають вирішальне значення для підтримки захисту та безпеки вашої системи ERP.

2.3.2 ERP інтеграція

Інтеграція ERP – це метод підключення системного програмного забезпечення ERP до інших систем. Інтеграція забезпечує потік інформації між різними системами. Це спосіб автоматизувати бізнес-процеси та підвищити продуктивність всього підприємства [4]. Це дозволяє підприємствам швидше обробляти замовлення та

збільшувати обробку замовлень на веб-сайті того ж дня, заощаджуючи багато грошей та часу. Без інтеграції з ERP адміністраторам довелося б вручну оновлювати рівні запасів у системі ERP і додатку електронної комерції. У міру збільшення обсягів продажу та зростання запасів неефективні та трудомісткі операційні процеси стають важко масштабованими. Інтеграція з ERP допоможе користувачам отримати огляд компанії та всіх запущених програм. Він допомагає відстежувати ключові показники ефективності в режимі реального часу, забезпечувати цілісність даних та багато іншого.

Роль систем ERP в системній інтеграції є вирішальною, оскільки вони містять дані про клієнтів, продукти та замовлення на продаж. Існує безліч типів ERP інтеграцій, які оптимізують бізнес-процеси, серед них основними є:

— HTTP(S) інтеграція;

— ESB інтеграція.

2.3.2.1 HTTP(S) інтеграція

HTTP(S) інтеграція з’єднує ERP систему із одним конкретним додатком (зазвичай – API сервером). Реалізовано це у вигляді звичайного HTTP (або HTTPS) запиту на вказану адресу із надсиланням даних. Дана інтеграція може бути використана як у системі декількох окремих додатків, розміщених на одному або різних серверах, так і для обміну даними серед додатками різних компаній (наприклад, ERP система інтернет-магазину може надіслати запит до API логістичної компанії для відправлення замовлення). Сам запит може бути налаштований (мати конкретний тип і/або заголовки запиту).

2.3.2.2 ESB інтеграція

Іншим рішенням є Enterprise Service Bus або ESB. Це локальна архітектура програмного забезпечення, яка дозволяє різним програмам спілкуватися одна з одною та обмінюватися даними [5]. Дані надходять на шину в певному форматі, часто JSON,

і існують адаптери, які знаходяться між шиною та ERP системою, яке перекладає їхні дані в/з JSON і в потрібний формат. Схема підключенням різних додатків до ESB шини ERP системи зображено на рисунку 2.2.

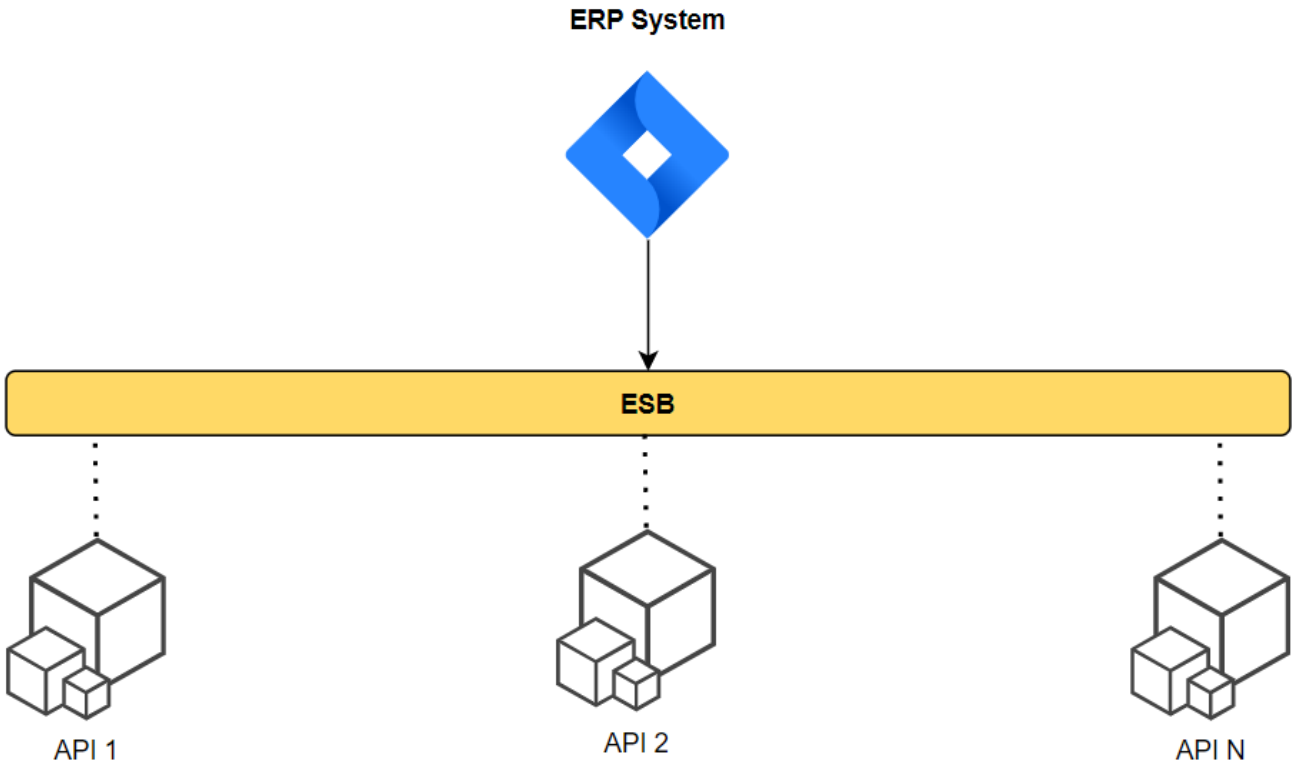

Рисунок 2.2 – Приклад ESB інтеграції

2.3.3 ERP IAM

Управління ідентифікацією та доступом (IAM) – це управління поточним життєвим циклом облікових записів та прав доступу до всіх корпоративних інформаційних ресурсів, як у корпоративних центрах обробки даних, так і у хмарі [6]. Ця функція є центральною для безпеки системи, оскільки вона автентифікує користувачів та регулює доступ до систем та даних. Підприємства можуть використовувати стратегію нульової довіри для визначення користувачів. Хмарні служби управління ідентифікацією використовують інтеграцію з відкритими стандартами зниження накладних витрат і витрат за обслуговування. Цей процес включає перевірку облікових даних користувача та відповідних системних прав

доступу. Рішення IAM надає інструменти для керування цифровою ідентифікацією користувачів системи та забезпечення належного доступу до ресурсів компанії. Ці рішення дозволяють адміністраторам відстежувати дії користувачів, створювати звіти про дії користувачів та застосовувати політики для забезпечення відповідності вимогам.

IAM є найважливішим інструментом захисту корпоративних ресурсів від загроз кібербезпеці. Система IAM забезпечує узгодженість правил та політик доступу користувачів у всій компанії, а також правильне та точне застосування прав доступу до ресурсів у міру того, як користувачі змінюють роль компанії. Без автоматизованого моніторингу ресурсів та дій підприємства вразливі до витоку облікових записів та даних. Це часто скомпрометовано проблемою надмірних дозволів, які не керуються належним чином.

Рішення IAM забезпечують керування доступом на основі ролей, що дозволяє адміністраторам регулювати доступ окремих користувачів до систем або мереж. Основною метою цього рішення є збір інформації про користувачів, управління ідентифікаційними даними користувачів та налаштування прав доступу з урахуванням життєвого циклу цих даних. Ключові функції включають:

– гарантії життєвого циклу облікових записів: управління життєвим циклом облікових записів користувачів, включаючи права та ініціалізацію;

– керування доступом: єдиний вхід (SSO) та багатофакторна автентифікація (MFA) часто використовуються для управління уніфікованими політиками доступу.

– служба каталогів: централізоване та уніфіковане керування та синхронізація даних облікового запису.

– ініціалізація користувачів: автоматичне створення та призначення нових облікових записів користувачів.

– аналіз ідентифікаційних даних: застосування машинного навчання для виявлення та запобігання підозрілим діям з використанням ідентифікаційних даних.

– єдиний вхід або єдиний вхід (SSO): об'єднання пароля та облікових даних користувача в один обліковий запис та використання надійних паролів для спрощення доступу до служб.

– багатофакторна аутентифікація (MFA): складніша аутентифікація у вигляді додаткових факторів контролю аутентифікації. Це допомагає гарантувати справжність користувача та знижує ризик крадіжки облікових даних.

– аутентифікація на основі ризику: використання алгоритмів розрахунку ризику дій користувача. Блокуйте та повідомляйте про ризиковані дії.

– контроль над ідентифікаційними даними та їхнє керування ними: зменшення ризику надмірного доступу та привілеїв за допомогою елементів керування привілеями.

2.4 Огляд існуючих рішень

На основі проведеного попереднього аналізу були відібрані найбільш популярні аналоги даного фреймворку.

2.4.1 Microsoft D365

Одна з основних причин, через яку D365 є найпопулярнішим аналогом, полягає в тому, що він пропонує два різні рішення. Є Business Central, створений для малого бізнесу, та Finance and Operations, призначений для більших і складніших організацій. [7].

Користувачі використовують дві абсолютно різні системи, що відповідають різним потребам різних типів організацій, але мають гнучкість і інтерфейс користувача Microsoft. Багатьом компаніям подобається цей інтерфейс, і вони цінують гнучкість, яку пропонує D365. Особливо в порівнянні з Oracle NetSuite або SAP S/4HANA, наприклад, Microsoft D365 є набагато простішим у формуванні.

Недоліком є те, що тільки те, що користувачі можуть змінити систему D365 не означає, що вони повинні це робити. Багато організацій настільки намагаються настроїти свої системи, що здаються під час впровадження. Ще одним привабливим аспектом Microsoft Dynamics є те, що її дуже легко інтегрувати з іншими системами.

На рисунку 2.3 можна побачити інтерфейс застосунку компанії Microsoft Business Central.

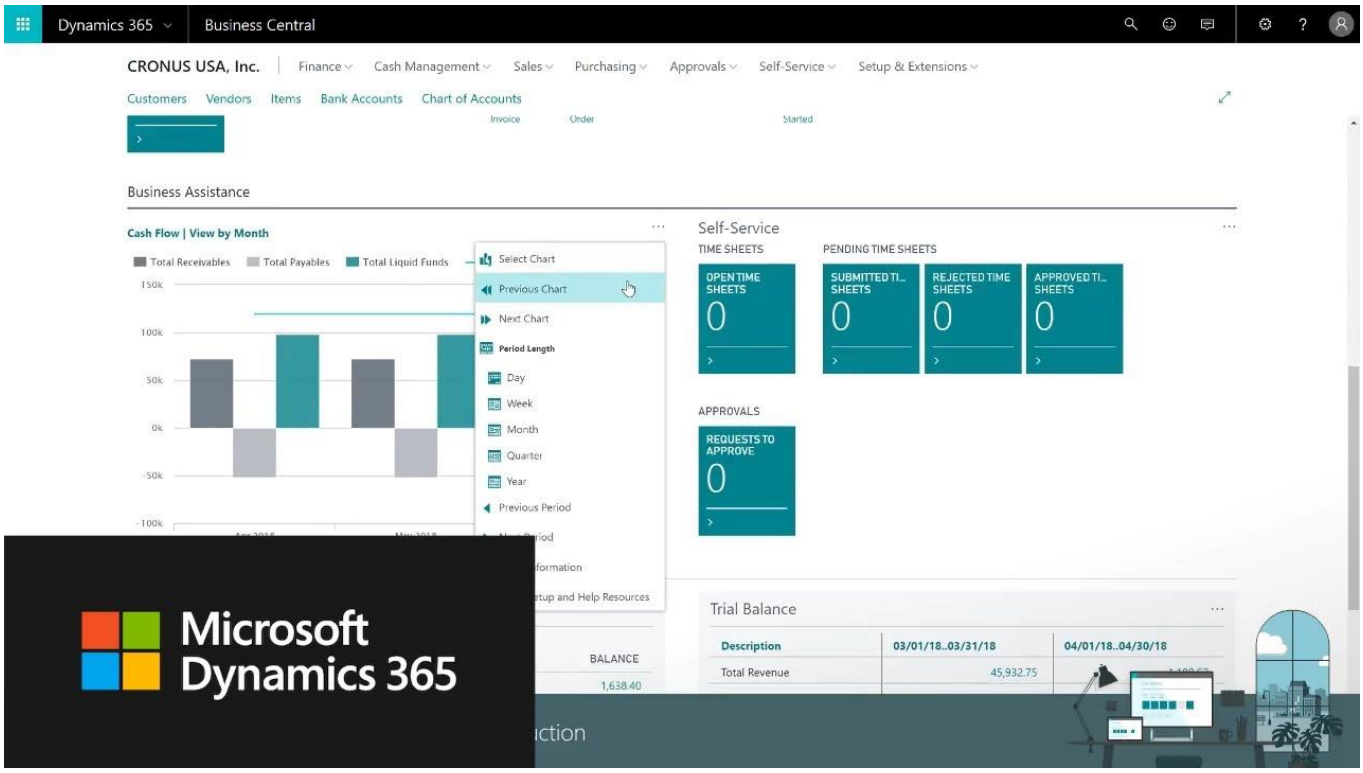

Рисунок 2.3 – Інтерфейс додатку MS Business Central

2.4.2 Oracle NetSuite

Позитивні аспекти Oracle NetSuite в тому, що це один із перших типів рішень «програмне забезпечення як послуга». Він знаходиться у хмарі вже 20 років, задовго до того, як його наздогнали решта постачальників. Таким чином, у них є дуже зріле рішення, яке все життя перебуває у хмарі. Він був створений для хмари та має хмарну архітектуру. NetSuite також орієнтований на малий бізнес, і в цьому випадку оновлення базової системи обліку до NetSuite може стати наступним логічним кроком у еволюції через цифрову трансформацію [8].

Недоліком Oracle NetSuite є досить висока ціна, особливо для малих і середніх організацій. Модель підписки, що повторюється, з великою кількістю прихованих витрат може з часом накопичуватися, тому в довгостроковій перспективі вона може

бути дуже дорогою. Що дійсно стримує NetSuite, так це відсутність гнучкості в порівнянні з іншими системами на ринку.

На рисунку 2.4 зображено інтерфейс додатку Oracle NetSuite.

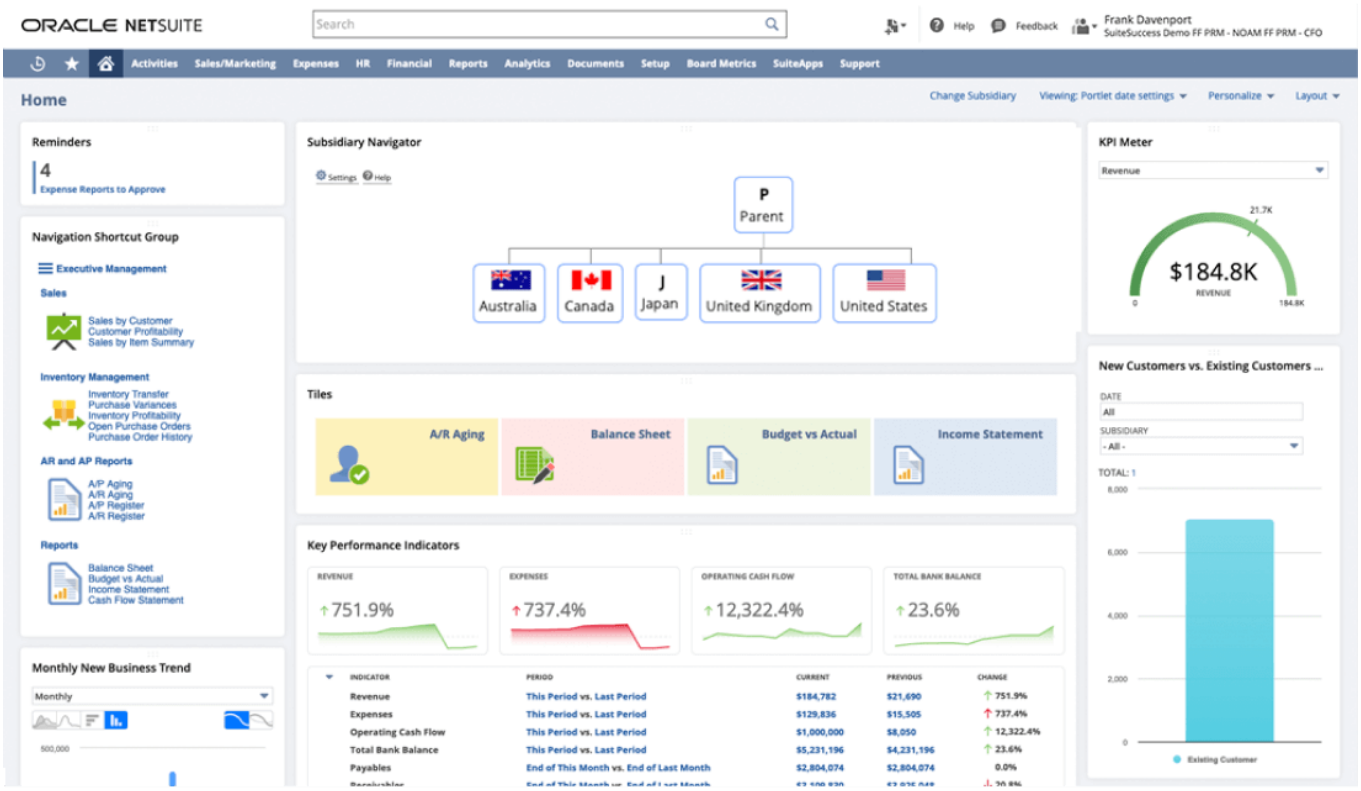

Рисунок 2.4 – Інтерфейс додатку Oracle NetSuite

2.4.3 Oracle ERP Cloud

Oracle ERP Cloud є одним із стандартів для великих організацій зі списку Fortune 1000. Oracle ERP Cloud – це дуже широкий і надійний продукт, який може задовольнити потреби багатьох галузей, особливо якщо для диверсифікованих, великих та складних організацій.

Oracle Cloud ERP – це повний, сучасний хмарний пакет ERP, який надає командам розширені можливості, такі як штучний інтелект для автоматизації ручних процесів, які сповільнюють їх роботу, аналітику для реагування на зміни ринку в режимі реального часу та автоматичні оновлення, щоб бути в курсі і отримати конкурентну перевагу [9].

На рисунку 2.5 зображено інтерфейс додатку Oracle ERP Cloud.

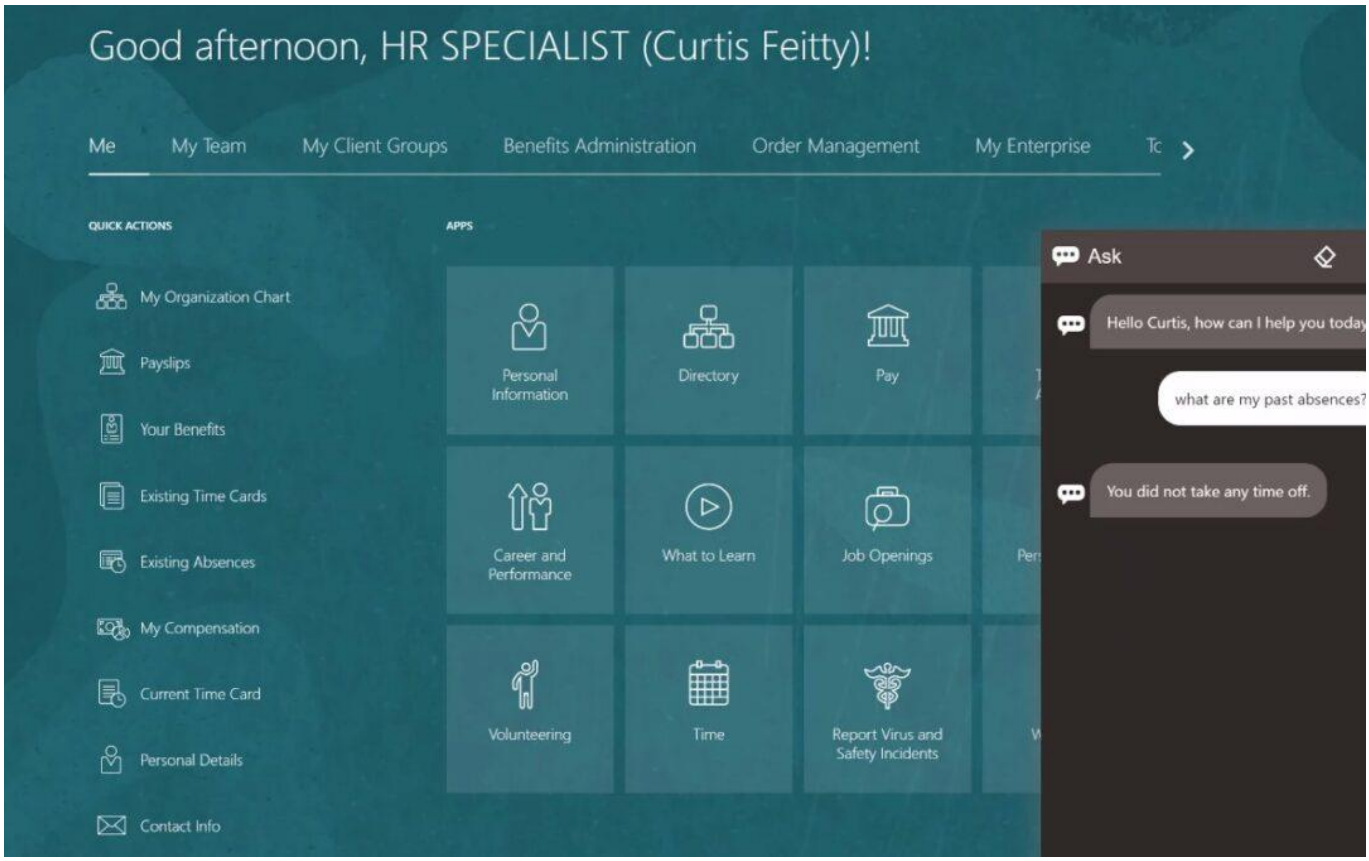

Рисунок 2.5 – Інтерфейс додатку Oracle ERP Cloud

2.4.4 Epicor Kinetic

Epicor Kinetic (раніше Epicor ERP) призначений для підтримки різних виробничих процесів, включаючи дискретні, виготовлення на замовлення (MTO), проєктування на замовлення (ETO), конфігурування на замовлення (CTO) та змішаний режим та середовище виготовлення на складі. Epicor продовжує надавати найкращі в галузі рішення новим, швидкозростаючим виробникам, а також компаніям середнього розміру та дочірнім компаніям великих транснаціональних корпорацій. Epicor ERP – це модульна, багатофункціональна, масштабована система, яка підтримує зростання компанії за рахунок швидкого впровадження та легкого розширення, незалежно від розміру чи складності виробничого процесу.

Це рішення пропонує повну гнучкість у розгортанні. Користувачі можуть вибрати розгортання рішення у хмарі або локально. Будучи багатовимірним рішенням, Epicor Kinetic for Manufacturing має унікальну здатність керувати вимогами кількох галузей за допомогою єдиного рішення, включаючи промислове обладнання, автомобілебудування, аерокосмічну та оборонну промисловість, медичні пристрої, електроніку та високі технології, готові метали, послуги з металообробки, меблі та гумові чи пластмасові пристрої, майстерні та тощо [10]. Epicor має досвід роботи у багатьох виробничих галузях та розробив рішення, що відповідають унікальним вимогам забезпечення надійної функціональності у цих галузях.

У міру того, як бізнес росте і змінюється, власникам необхідне рішення, здатне впоратися з цим зростанням та змінами. Epicor Kinetic пропонує велику гнучкість, оскільки його можна розгорнути локально або у хмарі. Наприклад, якщо у компанії обмежені IT-ресурси, користувачі можуть спочатку розгорнути рішення у хмарі. У міру зміни бізнесу користувачі можуть пізніше ухвалити рішення про повторне розгортання Epicor Kinetic On-Premises.

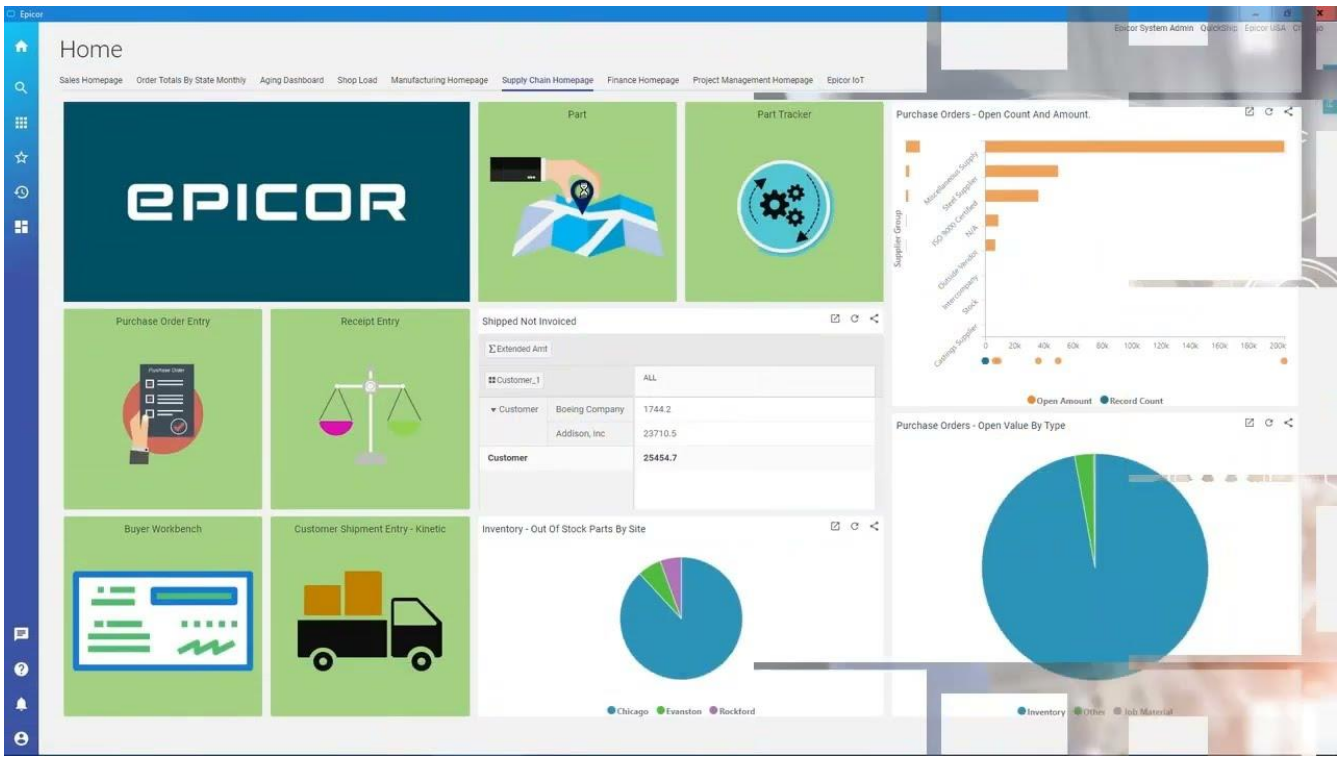

Рисунок 2.6 – Інтерфейс додатку Epicor Kinetic

2.4.5 SAP S/4HANA

S/4HANA відмінно підходить для фінансів, управління запасами, основних функцій ERP та багато іншого. Це просто один з кращих, коли йдеться про фінансову гнучкість, можливості GL, цінність продукту та багато іншого. З іншого боку, S/4HANA відсутні розширені функції, такі як виробництво, планування та управління життєвим циклом продукту, і навіть відсутні деякі функції CRM.

Що SAP зробила для часткового вирішення цієї проблеми, то це придбала інші компанії. Вони придбали такі продукти, як Ariba для закупівель, Success Factors для управління людським капіталом та Concur для економії часу та витрат. Вони стали певною мірою кращими у своєму класі постачальниками, але разом з цим виникає проблема інтеграції кількох систем [11]. Дорожня карта SAP все ще трохи незграбна, і важко зрозуміти, які продукти підходять для SAP.

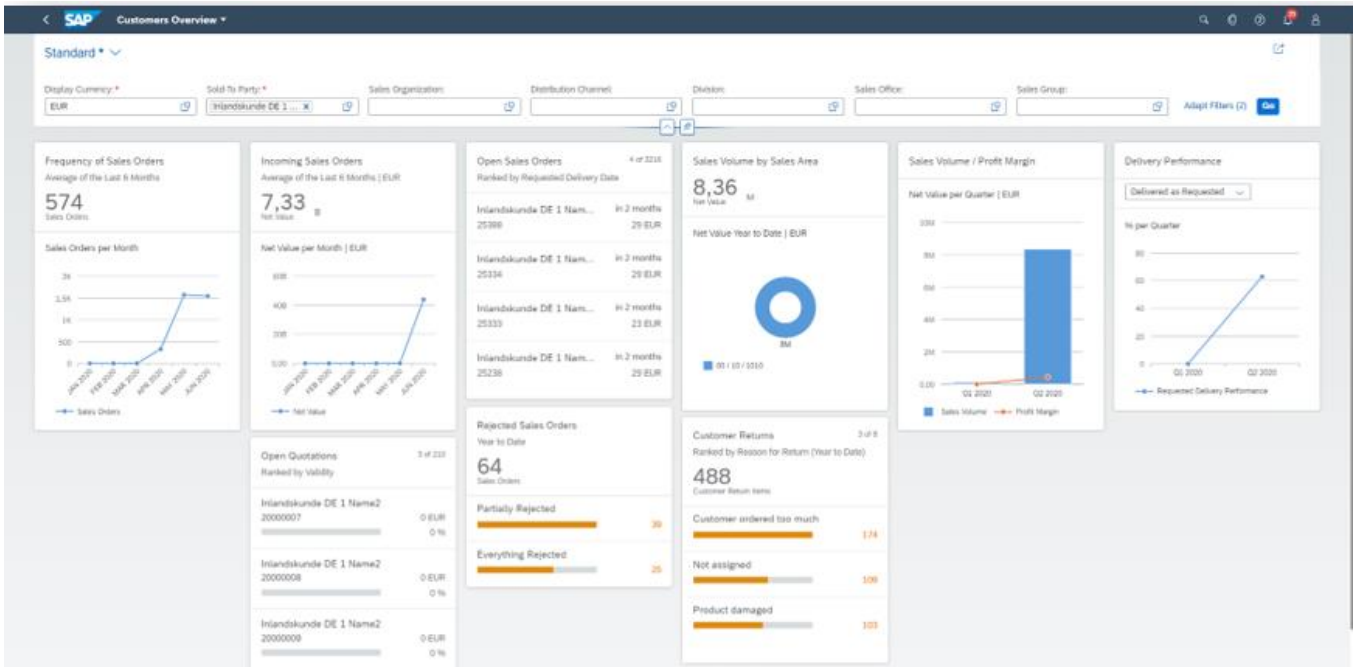

Рисунок 2.7 – Інтерфейс додатку SAP S/4HANA

Висновок до розділу 2

У цьому розділі були детально визначені основні компоненти ERP, які необхідно впровадити під час розробки цієї структури. Крім того, було детально досліджено концепцію модульності в рамках ERP-системи, розкрито деталі ідеї цієї концепції, підкреслено її сутність, обговорено її переваги як з точки зору архітектури фреймворку, так і з точки зору вигоди для бізнесу, особливо для власників даних. Ця концепція дає можливість додавати функціональні можливості у ERP-систему без впливу на інші компоненти функції. Додавання нового модуля до системи, видалення непотрібного модуля або виходу з ладу окремого модуля не впливає на роботу інших модулів або всієї системи, що полегшує розширення системи.

Також був проведений огляд існуючих аналогів фреймворку системи, що розглядається в даній роботі, в ході якого були виділені основні аспекти та особливості фреймворку, що розробляється. Це допомагає визначити технічні та функціональні вимоги.

3 ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ

На основі проведеного аналізу існуючих аналогів необхідно визначити детальні технічні та функціональні вимоги для проєктної реалізації фреймворку, що розглядається в даній роботі.

3.1 Технічні вимоги

3.1.1 Використання хмарних сервісів

Хмарні обчислення – це доставка IT-ресурсів на запит через Інтернет з оплатою за фактом використання. Замість того, щоб купувати, володіти та обслуговувати фізичні центри обробки даних та сервери, користувачі можуть споживати обчислювальну потужність, сховище, бази даних та інші технологічні послуги на запит від хмарних провайдерів.

Організації всіх типів, розмірів та галузей використовують хмару для різних цілей, включаючи резервне копіювання даних, аварійне відновлення, електронну пошту, віртуальні робочі столи, розробку та тестування програмного забезпечення, аналіз великих даних та клієнтські веб-додатки. Наприклад, медичні компанії використовують хмарні технології розробки більш персоналізованих методів лікування своїх пацієнтів. Фінансові компанії використовують хмару для виявлення та запобігання шахрайству в режимі реального часу.

Перевагами хмарних обчислень є наступні пункти:

– швидкість: хмара надає користувачеві легкий доступ до широкого спектру технологій, для швидшого впровадження інновацій;

– еластичність: у разі використання хмарних обчислень користувачам не потрібно попередньо виділяти ресурси для відповідності майбутнім піковим рівням ділової активності. Натомість вони отримують ту кількість ресурсів, яка їм справді потрібна. Клієнти можуть масштабувати ці ресурси, щоб миттєво збільшувати чи зменшувати ємність у міру зміни потреб бізнесу;

– економія коштів: у хмарі користувачі можуть замінити фіксовані витрати (наприклад, центри обробки даних та фізичні сервери) змінними витратами та платити за IT лише за те, що використовують. Крім того, змінні витрати набагато нижчі, ніж користувачі платять самі собі через ефект масштабу;

– вихід на глобальний рівень за лічені хвилини: хмара дозволяє користувачам освоювати нові географічні регіони та глобально розгортати програми та програми за лічені хвилини. Розміщення програм ближче до кінцевих користувачів скорочує час затримки та підвищує зручність роботи.

З урахуванням наведених вище аргументів, можна зробити висновок та прийняти за вимогу те, що усі додатки та функціонал сервісу має бути опублікований лише за допомогою хмарних технологій, що стане в нагоді в питанні економії часу і фінансів, а також полегшить процес управління архітектурою фреймворку в цілому.

3.1.2 Загальні технічні вимоги

Фреймворк, що розробляється, повинен давати користувачу можливість швидко створювати веб-сервіси ERP систем у глобальному інтернеті із глобальним інтернет доменом. Створити та видалити ERP системи повинна бути змога як і через головний веб-сайт, так і через запит до API даного фреймворку. При процесі створення користувач повинен мати досить різноманітний вибір налаштувань, таких як назва системи, тип бази даних, логін і пароль для доступу до БД тощо. Сам процес запуску ERP системи має займати невеликий проміжок часу – не більше години.

Основною вимогою до системи є концепція модульності. Система повинна мати можливість легко доповнюватись новими модулями, так само швидко і легко модулі мають видалятися із системи за необхідності. Самі модулі мають коректно працювати як в якості незалежного компоненту, так і у взаємодії із іншими компонентами системи в цілому.

Фреймворк має забезпечувати швидкий і безпечний процес передачі запитів та їхньої обробки. Цей процес має бути реалізований із використанням сучасних алгоритмів шифрування даних, таким як підпис запитів (англ. «request signing»).

У системі має бути ретельно реалізований процес ідентифікації користувача та його прав. Кожний запит від користувача повинен перевірятися із правами даного користувача та блокуватися у випадку недостатнього доступу.

Власник фреймворку повинен мати швидкий доступ до інформації даного сервісу у будь-який момент часу. Одним із варіантів вирішення даного питання є розробка окремого додатка конкретно для власника даного фреймворку.

Оскільки в перспективі даний сервіс може стати комерційним, і ERP системи клієнтів потребуватимуть фінансів для оплати хмарних ресурсів, важливо розробити механізм збереження інформації щодо використання користувачів тих або інших ресурсів, за які користувачеві надходитиме плата.

3.2 Функціональні вимоги

3.2.1 Вимоги до користувацьких можливостей головного веб-сайту

Головний веб-сайт є основним веб-застосунком, яким буде користуватися потенційний користувач фреймворку. До основних вимог щодо даного додатку належать:

– реєстрація за допомогою електронної пошти та паролю та верифікація шляхом надсилання електронного листа на вказану електронну пошту;

– авторизація за допомогою електронної пошти та паролю;

– можливість відновлення паролю через електронну пошту;

– можливість зміни паролю;

– перегляд створених ERP систем;

– створення ERP системи за бажаними налаштуванням та із бажаними доступними модулями;

– перегляд детальної інформації кожної ERP системи окремо;

– видалення ERP системи у будь-який момент після її повного запуску;

– перегляд рахунків та статистики використання ERP систем (кількість годин роботи тощо);

– ознайомлення із фреймворком в цілому та його конкретними можливостями;

3.2.2 Вимоги до користувацьких можливостей бек-офісу

Бек-офіс (англ. «back office», «BO») є веб-застосунком, розробленим виключно для власника фреймворку. Він слугуватиме для отримання власником актуальної інформації щодо сервісу. До основних вимог щодо даного додатку належать:

– перегляд зареєстрованих користувачів фреймворку;

– перегляд створених ERP систем;

– перегляд існуючих модулів, додавання нових, а також оновлення та видалення існуючих;

– перегляд статусу працездатності інших застосунків або ресурсів фреймворку, а саме: головного веб-сайту, головної бази даних та API;

– перегляд статусу використання хмарних ресурсів, таких як обчислювальні машини, бази даних, інтернет доменів, хмарне сховище тощо.

3.2.3 Вимоги до користувацького процесу веб-інтерфейсу ERP системи

Веб-інтерфейс ERP системи також є не менш важливим, оскільки саме цей застосунок користувач побачить в якості кінцевого результату, і саме із цим додатком він буде працювати. До основних вимог щодо даного застосунку належать:

– перегляд встановлених модулів та їхнє видалення із ERP системи;

– доступ лише для власника ERP системи;

– унеможливлення неавторизованому користувачу переходу на всі сторінки, крім сторінки авторизації;

– перегляд даних кожних таблиць встановлених модулів;

– запис у системний журнал усієї активності та запитів від користувачів;

– перегляд системного журналу;

– створення, редагування та видалення IAM облікових записів;

– створення, редагування та видалення інтеграцій та їхній статус;

– перегляд налаштувань ERP системи;

– перегляд статусу мережі та статистики запитів;

– забезпечення швидкої і надійної обробки запитів від користувача та повідомляти про помилки під час обробки;

– перегляд доступних невстановлених модулів та їхня інсталяція до системи.

3.2.4 Вимоги до користувацьких можливостей консолі взаємодії та управління

Консоль управління є веб-застосунком для працівників компанії. Саме через цей застосунок працівники підприємства будуть авторизуватись, використовуючи ключі доступу IAM облікових записів, і працювати із ERP системою. Основними вимогами до цього додатку є наступні пункти:

– унеможливити неавторизованим користувачам переходити на будь-які сторінки додатку, окрім сторінки авторизації;

– візуальне відображення та доступний функціонал має повністю залежати від прав доступу авторизованого користувача;

– стабільна система запитів до ERP системи користувача.

3.2.5 Вимоги до процесу роботи API

API даного фреймворку повинен надавати можливість отримувати інформацію для користувачів не лише через головний веб-сайт, а і через звичайні запити. Це дасть можливість іншим розробникам інтегрувати свої застосунки із даним фреймворком та отримувати необхідні дані. До основних вимог щодо даного застосунку належать:

– обробка запитів лише для зареєстрованих користувачів, що пройшли верифікацію;

– система запитів має бути захищена за існуючими стандартами ERP безпеки;

– запити мають оброблятися швидко, не залежачи від кількості трафіку.

3.2.6 Вимоги до розробки пакету інструментів розробки SDK

Останньою вимогою є створення пакету інструментів розробки (англ. «Software Development Kit»). SDK покращують стандартні процеси та надають доступ до необхідної інформації та створені для інформування, надання необхідних інструментів і шаблонів, завдяки чому розробники можуть сфокусуватися на розробці свого продукту [12].

Висновок до розділу 3

У цьому розділі був визначений список основних технічних та функціональних вимог фреймворку, що розглядається у цій роботі. Підставою для цих вимог є дані, одержані на основі огляду та аналізу існуючих рішень.

Використання хмарних сервісів може заощадити багато фінансових ресурсів та значно прискорити процес розробки та впровадження нових функцій.

Запропоновано технічну вимогу щодо підписання запитів для забезпечення надійності захисту передачі інформації між додатками. Оскільки йдеться про корпоративні дані, крадіжка цієї інформації може призвести до значних фінансових втрат для компанії.

Вимоги до модульності реалізовані, щоб клієнти могли додавати необхідні функції до своєї ERP-системи відповідно до потреб свого бізнесу.

4 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

На основі визначених технічних та функціональних вимог потрібно провести аналіз технологічного стеку, принципів побудови веб-застосунків та вибрати найбільш сучасні технології, які стануть в нагоді при розробці.

4.1 Мова програмування C# та платформа .NET

В якості головної технології розробки веб-застосунків була вибрана платформа .NET [13] від компанії Microsoft та її головна мова програмування C#.

Основна перевага полягає в тому, що C# – це сучасна об'єктно-орієнтована мова програмування, також від Microsoft [14]. Ця мова є наступником мови C++, включає статичну типізацію даних та підтримує принципи ООП. Основні особливості цієї мови програмування::

– об'єктно-орієнтована, компонентно-орієнтована мова програмування;

– механізм складання сміття автоматично звільняє пам'ять, зайняту об'єктами, що не використовуються;

– обробка винятків забезпечує структурований та розширюваний підхід до виявлення та усунення помилок;

– синтаксис Language Integrated Query (LINQ) створює загальний шаблон для керування даними з будь-якого джерела;

– мовна підтримка асинхронних операцій забезпечує синтаксис для побудови розподілених систем;

– підтримка механізмів, що використовують залежність (Dependency Injection);

– C# підтримує керування версіями, тому програми та бібліотеки можуть згодом розвиватися сумісним чином.

4.2 Blazor

В якості технології створення веб-застосунків платформи .NET був обраний сучасний механізм під назвою Blazor. Даний інструментарій на основі ASP.NET Core активно розвивається компанією Microsoft, є дуже популярним серед .NET розробників завдяки легкості роботи і стає все більш популярним [15].

Дана технологія по своїй суті є аналогом SPA від компанії Microsoft. SPA (Single Page Application) є сучасним типом веб-додатків, який є односторінковим додатком, яке завантажується на одну HTML сторінку, і далі, завдяки механізму мови JavaScript, динамічно оновлює контент сторінки, завантажуючи потрібні компоненти.

Перевагою такого типу є те, що в процесі роботи користувачеві може здатися, що він запустив не веб-сайт, а десктопний додаток, оскільки він миттєво реагує на всі його дії, без затримок та «підвисань», також такий додаток має вбудовану оптимізацію та розподіл ресурсів, що суттєво зменшить навантаження.

Основними фреймворками такого типу веб-застосунків саме на мові JS є:

– Angular;
– React;
– Vue JS.

Єдиною відмінністю між переліченими фреймворками та інструментом Blazor є те, що в Blazor при розробці застосунків використовується мова C#, а не JS, що полегшує роботу .NET розробникам, оскільки більше немає потреби вивчати JS в якості додаткової мови програмування, хоча навіть у випадку необхідності використання саме цієї мови (наприклад, для графічних задач), у Blazor є вбудований механізм виклику JS скриптів.

4.2.1 Blazor Server

На даний момент, існує 2 типи веб-застосунків фреймворку Blazor:

– Blazor Server;
– Blazor WebAssembly.

Різниця між вказаними типами в тому, що у Blazor Server зміни у UI фіксуються та обробляються механізмом SignalR, і надсилаються на сервер, на стороні якого генерується оновлений UI і повертається клієнтові (рисунок 4.1), в той час як у Blazor WebAssembly маніпуляції із UI відбуваються за допомогою вищезгаданої мови JavaScript.

WebAssembly за своєю суттю є PWA – прогресивна програма або Progressive Web Application, яка взаємодіє із користувачем, як додаток. Вона може встановлюватися на головний екран смартфона, надсилати push-сповіщення та працювати в офлайн-режимі. Перевагами такого типу є кросплатформенність, висока швидкість роботи та можливість запуску та відображення даних в офлайн-режимі з моментальним завантаженням та висока швидкість встановлення.

Важливою перевагою Server над WebAssembly є те, що у випадку із WebAssembly бібліотеки із згенерованим програмним кодом будуть доступні клієнтові, оскільки, як вже було визначено, обробка запитів відбувається саме на стороні клієнта. Це означає, що клієнт у будь-який момент може в налаштуваннях браузера, у секції Developer Tools, знайти ці бібліотеки та скачати вихідний код додатку, що є негативним аспектом із точки зору безпеки.

Сам через цю причину перевага буде надана саме Blazor Server.

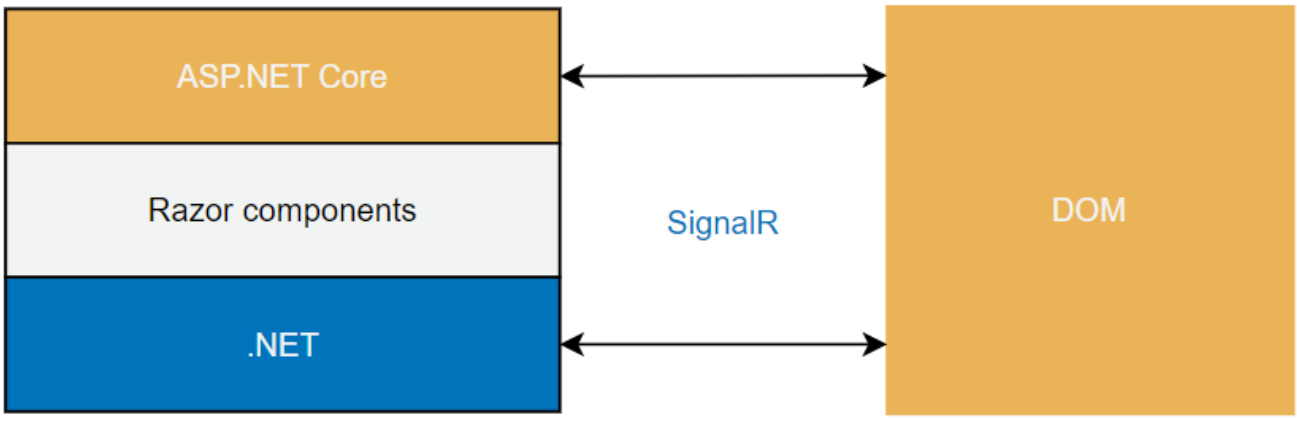

Рисунок 4.1 – Схема взаємодії UI та серверу у Blazor Server

4.2.2 SignalR

SignalR ASP.NET Core — це бібліотека з відкритим кодом, яка спрощує створення веб-функцій у режимі реального часу до програм [16]. Веб-функції в режимі реального часу дозволяють із серверу миттєво надсилати вміст клієнтам. SignalR надає API для створення викликів віддалених процедур між серверами (RPC). RPC викликають функції на клієнтах із коду .NET Core на стороні сервера. Існує кілька платформ, що підтримуються, кожна з яких має відповідний клієнтський пакет SDK. Через це виклик RPC викликає різні мови програмування.

До основних функцій SignalR належать:

– автоматично обробляє керування підключеннями;

– одночасно надсилає повідомлення всім підключеним клієнтам;

– надсилає повідомлення певним клієнтам чи групам клієнтів (рисунок 4.2).

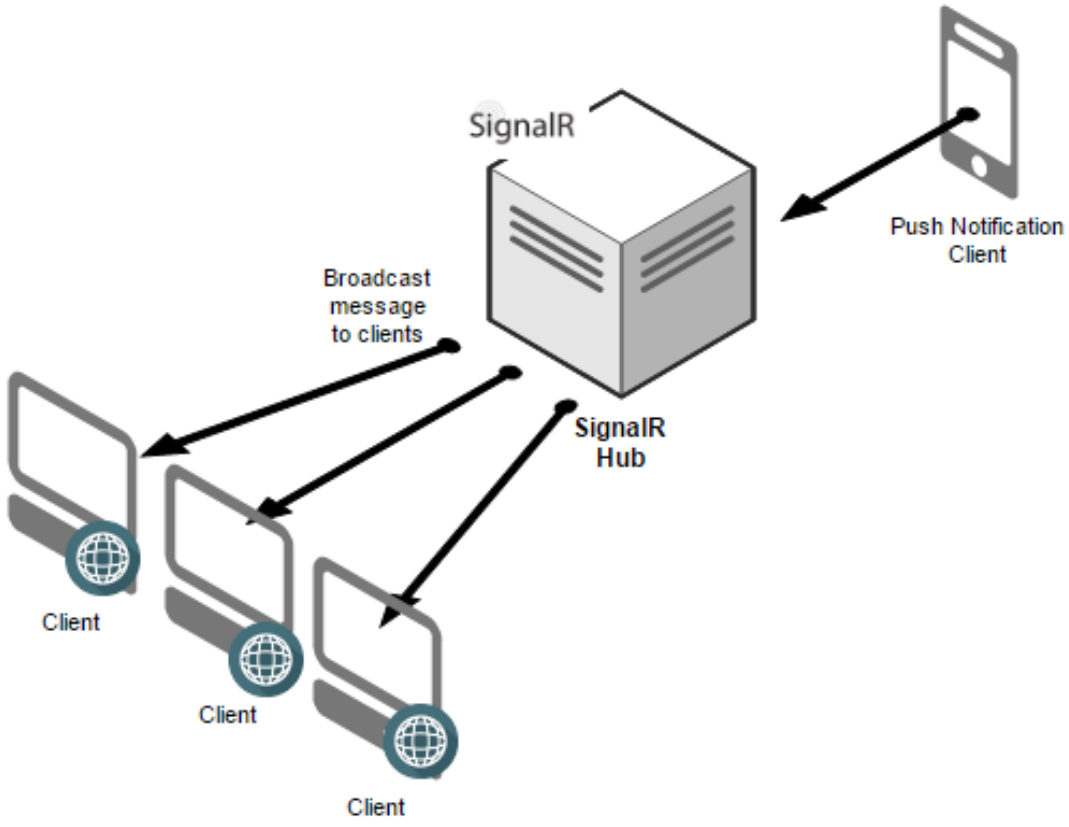

Рисунок 4.2 – Схема прикладу роботи SignalR

4.3 Entity Framework Core

Як і у будь-якому застосунку, даний фреймворк повинен мати механізм роботи із базою даних. В якості такого інструменту був вибраний Entity Framework Core.

Entity Framework (EF) Core – це полегшена, розширювана, крос-платформна версія популярної технології доступу до даних Entity Framework з відкритим вихідним кодом. EF Core також працює як об'єктно-реляційне зіставлення (ORM) [17], який:

– дозволяє розробникам .NET працювати з базою даних за допомогою .NET об’єктів;

– усуває потребу в більшості коду доступу до даних, який зазвичай потрібно писати.

EF Core використовує моделі для доступу до даних. Модель складається з класів сутностей та контексту, що представляє сеанс із базою даних. Саме через контекст відбуваються запити на читання та збереження даних (рисунок 4.3).

У EF Core успішно абстрагується від більшої частини деталей програмування, але є деякі рекомендації, які застосовуються до будь-якої ORM, які допомагають уникнути поширених помилок у робочих додатках:

– базові знання сервера баз даних середнього чи просунутого рівня необхідні проектування, налаштування, профілювання та перенесення даних у високопродуктивних виробничих додатках. Наприклад, знання первинних та зовнішніх ключів, обмежень, індексів, нормалізації, операторів DML та DDL, типів даних, профілювання тощо;

– тест продуктивності. Звичайне використання деяких функцій погано масштабується. Наприклад, використання кількох колекцій, інтенсивне використання відкладеного завантаження (англ. «lazy loading»), умовні запити до неіндексованих стовпців, масові оновлення та вставки з використанням згенерованих значень, відсутність паралелізму, великі моделі та погана політика кешування тощо.

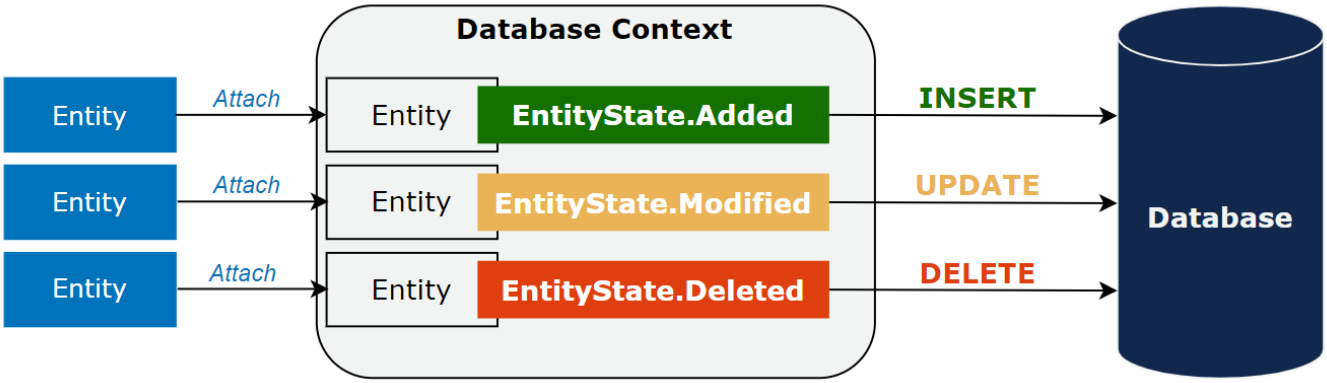

Рисунок 4.3 – Процес роботи із БД через EF Core

4.4 Шаблон проєктування CQRS

CQRS інтерпретується як Розподіл відповідальності команд та запитів (англ. «Command and Query Responsibility Segregation») [18], шаблон, який поділяє операції читання та оновлення сховища даних. Реалізація CQRS у додатку може максимізувати його продуктивність, масштабованість та безпеку. Гнучкість, створена переходом на CQRS, дозволяє системі краще розвиватися з часом і уникати виклику команд оновлення конфліктів злиття лише на рівні домену.

4.4.1 Контекст та суть шаблону

Традиційні архітектури використовували одну й ту саму модель даних для запитів та оновлень бази даних. Це підходить для основних операцій CRUD (створення, читання, оновлення, видалення). Однак у складніших додатках цей підхід може стати громіздким. Наприклад, на стороні читання програма може робити безліч різних запитів, які повертають об'єкти передачі даних (DTO) різних форматів. Перегляд об'єктів може бути ускладненим. На боці запису моделі можуть реалізовувати складну перевірку та бізнес-логіку. В результаті можна отримати надмірно складні моделі.

Робочі навантаження читання та запису часто асиметричні, з дуже різними вимогами до продуктивності та масштабування.

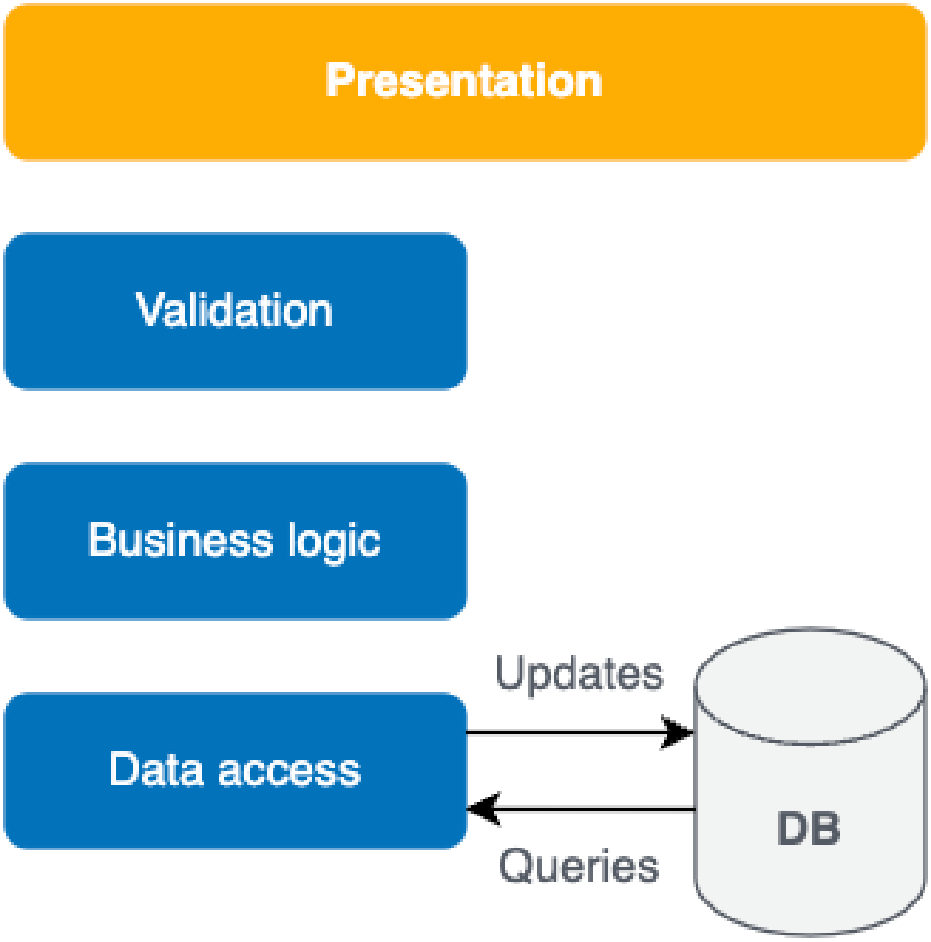

Рисунок 4.4 – Схема традиційної CRUD архітектури

До недоліків цієї архітектури можна віднести такі аспекти:

– часто виникає конфлікт між прочитаним та записаним поданням даних. Наприклад, додаткові стовпці або властивості, що не беруть участі в операції, але потребують коректного оновлення;

– потенційні гонки даних, коли операції виконуються паралельно одному наборі даних;

– традиційні підходи можуть негативно позначитися на продуктивності через навантаження на сховище даних та рівень доступу до даних, а також складність запитів, необхідних для отримання інформації;

– управління безпекою та доступом може бути складним, оскільки кожен об'єкт піддається операціям читання та запису, а дані можуть бути подані у неправильному контексті. CQRS поділяє операції читання та запису на різні моделі, використовуючи команди для оновлення даних та запити на читання даних;

– команди мають ґрунтуватися на завданнях, а не на базі даних. («Забронювати номер у готелі» замість «Встановити статус бронювання «Зарезервовано»);

– команди можна ставити у чергу для асинхронної обробки замість синхронної;

– запити ніколи не змінюють бази даних. Запит повертає DTO, які не інкапсулюють дані домену.

Наявність окремих моделей запитів та оновлень спрощує проектування та реалізацію. Однак одним недоліком є те, що код CQRS не може бути автоматично згенерований із схеми бази даних із використанням механізмів інфраструктури, таких як інструменти O/RM.

Для більшої ізоляції читання можуть бути фізично відокремлені від даних запису. У цьому випадку база даних, що зчитує, може використовувати власну схему даних, оптимізовану для запитів. Наприклад, ви можете зберігати матеріалізовані уявлення ваших даних, щоб уникнути складних об'єднань та складних відображень O/RM. Ви також можете використовувати різні типи сховищ даних. Наприклад, база даних запису може бути реляційною, а база даних читання – базою даних документів.

Якщо використовується окремі бази даних для читання та запису, їх слід синхронізувати. Зазвичай це досягається шляхом генерації події щоразу, коли модель запису оновлює базу даних. Брокери повідомлень та бази даних зазвичай не можуть бути частиною однієї розподіленої транзакції, що може призвести до проблем із забезпеченням узгодженості при оновленні баз даних та публікації подій.

Сховище для читання може бути доступною тільки для читання копією сховища для запису, або сховище для читання та запису може мати різні структури. Використання кількох реплік лише для читання може підвищити продуктивність запитів, особливо у розподілених сценаріях, коли репліки лише з читання розташовані поруч із екземплярами додатків.

Поділ сховища для читання та запису також дозволяє відповідним чином масштабувати кожну з них із навантаженням. Наприклад, сховище для читання зазвичай набагато дорожче, ніж сховище для запису.

4.4.2 MediatR

MediatR є бібліотекою для мови програмування C#, завдяки можна дуже просто та швидко інтегрувати шаблон CQRS у додаток [19]. На рівні коду даний шаблон реалізується двома взаємопов'язаними класами: один виступає в якості моделі запиту (реалізацією інтерфейсу IRequest<>, який вказує, який тип даних дає повертати даний запит) та обробника даного запиту (реалізації інтерфейсу IRequestHandler<>), який має реалізувати метод Handle, параметром якого є конкретний об'єкт запиту (рисунок 4.5).

```
public class MediatrExampleRequest : IRequest<string>
{
    public int Property { get; set; }
}

public class MediatrExampleRequestHandler : IRequestHandler<MediatrExampleRequest, string>
{
    public Task<string> Handle(MediatrExampleRequest request, CancellationToken cancellationToken)
    {
        // Business logic
        throw new NotImplementedException();
    }
}
```

Рисунок 4.5 – Приклад використання MediatR для реалізації шаблону CQRS

Після створення запиту та обробника, достатньо просто надіслати даний запит, використовуючи інтерфейс IMediator, попередньо додавши його до колекції сервісів механізму DI (Dependency Injection) (рисунок 4.6 – 4.7).

```
public static IServiceCollection AddMediatR(this IServiceCollection services)
{
    services.AddMediatR(Assembly.GetExecutingAssembly());

    return services;
}
```

Рисунок 4.6 – Використання IMediator у механізмі DI.

```
public class MediatrExampleService
{
    private readonly IMediator _mediator;
    public MediatrExampleService(IMediator mediator)
    {
        _mediator = mediator;
    }

    public async Task SomeMethod()
    {
        var request = new MediatrExampleRequest
        {
            Property = 1
        };

        var result = await _mediator.Send(request);
    }
}
```

Рисунок 4.7 – Приклад виклику запиту через механізм MediatR.

Важливо вказати, що у кожного запиту має бути лише один обробник.

Більш того, можна надсилати запити під час обробки цих самих запитів. Приклад даного функціоналу зображено на рисунку 4.8.

```
public class MediatrExampleRequest1 : IRequest<int>
{
    public int MyProperty { get; set; }
}

public class MediatrExampleRequestHandler1 : IRequestHandler<MediatrExampleRequest1, int>
{
    public async Task<int> Handle(MediatrExampleRequest1 request, CancellationToken cancellationToken)
    {
        // Business logic
        return await Task.FromResult(1);
    }
}

public class MediatrExampleRequest2 : IRequest<int>
{
    public int MyProperty { get; set; }

}
public class MediatrExampleRequestHandler2 : IRequestHandler<MediatrExampleRequest2, int>
{
    private readonly IMediator _mediator;

    public MediatrExampleRequestHandler2(IMediator mediator)
    {
        _mediator = mediator;
    }

    public async Task<int> Handle(MediatrExampleRequest2 request, CancellationToken cancellationToken)
    {
        var request1 = new MediatrExampleRequest1
        {
            MyProperty = 1
        };

        var requestiResult = await _mediator.Send(request1, cancellationToken);
        return requestiResult + 1;
    }
}
```

Рисунок 4.8 – Приклад використання IMediator в обробниках запитів

4.5 PostgreSQL

Існує безліч систем управління реляційними базами даних (RDBMS) [20], з яких можна вибирати, якщо реляційна модель найкраще представляє дані. PostgreSQL – одна з найпопулярніших і найпопулярніших у світі реляційних баз даних з відкритим вихідним кодом [21].

Одна з найбільш фундаментальних відмінностей між PostgreSQL та більшістю інших реляційних баз даних – це фундаментальна структура.

Більшість реляційних баз даних найкраще описати як РСУБД. СУБД - це програмне забезпечення, спеціально розроблене для роботи з реляційними базами даних, де дані зберігаються в табличних структурах з певними стовпцями та типами даних. Дані можна вимагати, змінювати та вилучати за допомогою методів, заснованих на реляційній алгебрі, зазвичай з використанням мови структурованих запитів (SQL).

PostgreSQL є технічно системою управління об'єктно-реляційними базами даних (ORDBMS). Це означає, що вона має самі реляційні функції, як і РСУБД, але й деякі об'єктно-орієнтовані функції.

Практично кажучи, це означає, що PostgreSQL дозволяє:

– визначати власні складні типи даних;
– перевантажувати функції для роботи з різними типами даних аргументів;
– визначати відносини успадкування між таблицями.

Ці функції є потужними інструментами, які допомагають працювати з базами даних та даними, використовуючи ті ж методи, які використовуються під час програмування. Підвищена гнучкість дозволяє моделювати різні типи та відносини всередині системи баз даних, а не поза програмою. Це дозволяє підтримувати узгодженість та наближати передбачувану поведінку до реальних даних.

PostgreSQL відомий тим, що надає відмінні реалізації основних реляційних функцій, не обмежуючись традиційними середовищами СУБД. Жодна база даних не може задовольнити всі потреби, але PostgreSQL — відмінний варіант завдяки своїй універсальності для багатьох випадків використання.

4.6 Вибір постачальника хмарних послуг

Як було зазначено у функціональних вимогах, даний сервіс має бути повністю опублікований із використанням хмарних технологій. Виходячи з цього, необхідно зробити аналіз найбільш популярних постачальників хмарних послуг, щоб вибрати найбільш надійного. У даному аналізі мають бути враховані такі показники як різноманіття сервісів, доступність пакету інструментів розробки (SDK) для вибраної мови програмування та ціна використання сервісів.

До переліку обов'язкових сервісів, які має надавати обраний провайдер, належать наступні:

– хмарне сховище об'єктів;
– сервіс баз даних;
– сервіс балансування навантажень;
– сервіс електронної пошти;
– безсерверні обчислення;
– сервіс реєстрації доменів.

4.6.1 Amazon Web Services

Amazon Web Services (AWS) – це комерційна загальнодоступна хмара, що підтримується і розробляється Amazon з 2006 року. Абонентам надаються послуги як на основі інфраструктурної моделі (віртуальні сервери, ресурси зберігання), так і рівня платформи (хмарні бази даних, хмарне ПЗ, хмарні безсерверні обчислення, засоби розробки). AWS — найпопулярніша у світі хмарна платформа, що пропонує понад 200 повнофункціональних сервісів із центрів обробки даних по всьому світу. AWS використовують мільйони клієнтів, у тому числі найшвидші стартапи, найбільші підприємства та провідні державні установи [22].

Gartner Research позиціонує AWS у квадранті лідерів нового Магічного квадранту хмарної інфраструктури та платформних послуг (англ. CIPS) 2021 року. CIPS, у контексті цього магічного квадранта, визначаються як «стандартизовані

високоавтоматизовані пропозиції, у яких ресурси інфраструктури (наприклад, обчислення, мережа та зберігання) доповнюються інтегрованими сервісами платформи» (рисунок 4.9).

Рисунок 4.9 – Квадрант хмарної інфраструктури та платформних послуг

Переваги AWS як постачальника хмарних послуг:

– максимальна функціональність. Від інфраструктурних технологій, таких як обчислення, зберігання та бази даних, до нових технологій, таких як машинне навчання та штучний інтелект, озера даних та аналітика, до Інтернету речей, AWS пропонує набагато більше послуг та можливостей, ніж будь-який інший постачальник хмарних послуг;

– найбільша спільнота клієнтів та партнерів AWS має найбільшу та динамічну спільноту з мільйонами активних клієнтів та десятками тисяч партнерів по всьому світу. Клієнти майже всіх галузей і розмірів, включаючи стартапи, підприємства та організації державного сектора, використовують AWS для різних сценаріїв використання. У партнерську мережу AWS входять тисячі системних інтеграторів, що спеціалізуються на сервісах AWS, та десятки тисяч незалежних постачальників програмного забезпечення, які адаптують свої технології до роботи на AWS;

— AWS розроблена як найгнучкіше і найбезпечніше середовище хмарних обчислень, доступне на сьогоднішній день. Наша основна інфраструктура призначена для задоволення вимог безпеки військових, міжнародних банків та інших важливих організацій. Це підтримується широким набором інструментів хмарної безпеки з більш ніж 300 службами та можливостями безпеки, відповідності та управління. AWS підтримує 98 стандартів безпеки та сертифікатів відповідності, і всі 117 сервісів AWS, де зберігаються дані клієнтів, пропонують можливість шифрування цих даних;

– найбільш оперативний досвід. AWS має неперевершений послужний список, надійність, безпеку та продуктивність, на які можна покластися при роботі з найбільш важливими програмами. Вже понад 16 років AWS надає хмарні послуги для різних варіантів використання мільйонам клієнтів по всьому світу. З усіх хмарних провайдерів AWS має найбільший досвід у масштабуванні;

– глобальна мережа регіонів. AWS має найбільш розгалужену глобальну хмарну інфраструктуру. Модель регіону та зони доступності AWS визнана Gartner рекомендованим підходом для запуску корпоративних програм, що потребують високої доступності.

Окрім цього, AWS надає різноманітний набір пакетів інструментів розробки SDK, в тому числі і для обраної мови програмування C#. AWS SDK для .NET спрощує використання служб AWS, надаючи набір узгоджених і звичних для розробників .NET бібліотек. Усі AWS SDK забезпечують підтримку розгляду життєвого циклу API, такого як керування обліковими даними, повторні спроби, маршалінг даних і серіалізація. Зовнішній вигляд UI консолі управління AWS зображено на рисунку 4.10.

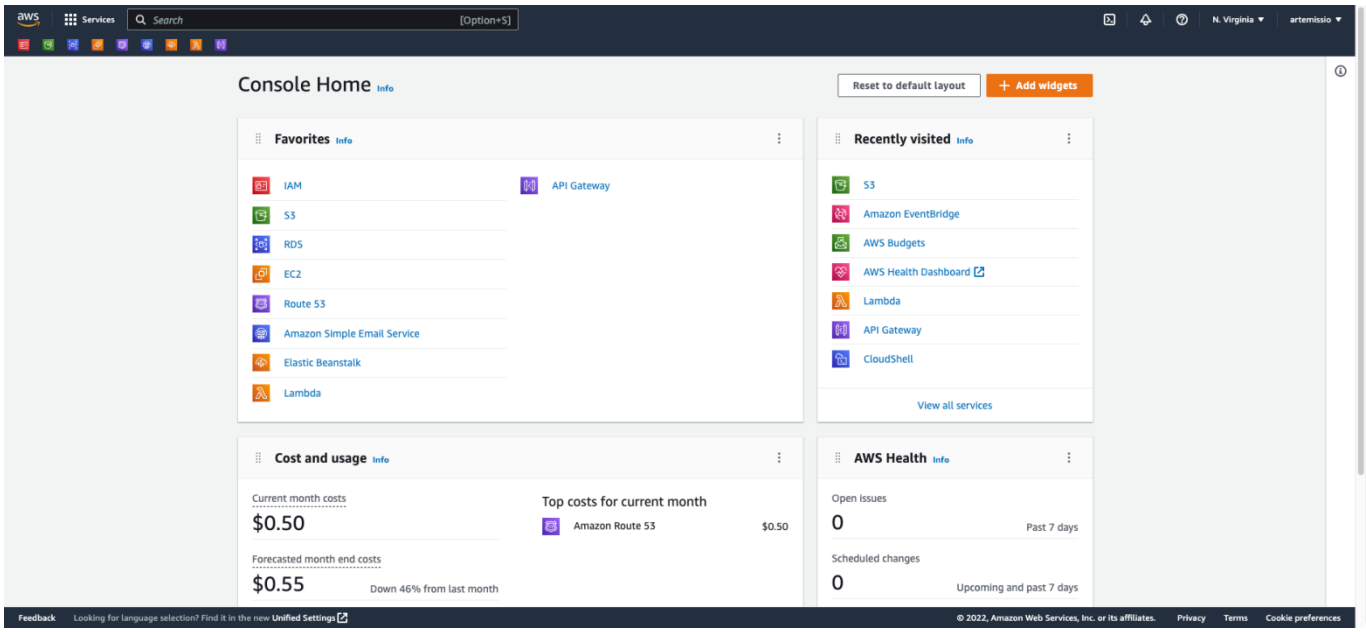

Рисунок 4.10 – UI консолі AWS Management Console

Ще однією особливістю AWS є наявність безкоштовного пробного періоду впродовж року з моменту реєстрації облікового запису. Доступні три різні типи безкоштовних пропозицій залежно від використовуваного продукту.

Усі необхідні для розробки фреймворку хмарні сервіси присутні у даного провайдера.

4.6.2 Google Cloud Platform

Google Cloud Platform (GCP) — це набір хмарних сервісів, які пропонують Google, які працюють на тій самій інфраструктурі, яку Google використовує для споживчих продуктів, таких як Google Search та YouTube. Крім інструментів управління, він також пропонує ряд модульних хмарних сервісів, таких як хмарні обчислення, зберігання даних, аналіз даних та машинне навчання [23].

Google Cloud складається з набору фізичних активів, таких як комп'ютери та жорсткі диски, та віртуальних ресурсів, таких як віртуальні машини (ВМ), розташовані в центрах обробки даних Google по всьому світу. Кожен центр обробки даних знаходиться у межах регіону. Доступні регіони в Азії, Австралії, Європі,

Північній Америці та Південній Америці. Кожен регіон є сукупністю зон, ізольованих одне від одного всередині регіону.

Такий розподіл ресурсів має кілька переваг, таких як відмово-стійкість та зменшення затримки за рахунок розміщення ресурсів ближче до клієнтів. Ця квота також запроваджує деякі правила спільного використання ресурсів.

Google Cloud надає клієнтські бібліотеки, які спрощують створення ресурсів та керування ними. Клієнтські бібліотеки Google Cloud надають API переважно для двох цілей:

– API-інтерфейси програм надають доступ до служб. API програми оптимізовано для мов, що підтримуються, таких як Node.js, Python і C#. Бібліотека заснована на метафорі служби, дозволяючи користувачам працювати зі службами природніше і писати менше стандартного коду. Бібліотека також надає помічників для автентифікації та авторизації;

– API керування надає функціональні можливості для керування ресурсами. Наприклад, якщо потрібно створити власні інструменти автоматизації, є можливість використовувати API управління.

Зовнішній вигляд UI консолі GCP Cloud Console зображено на рисунку 4.11.

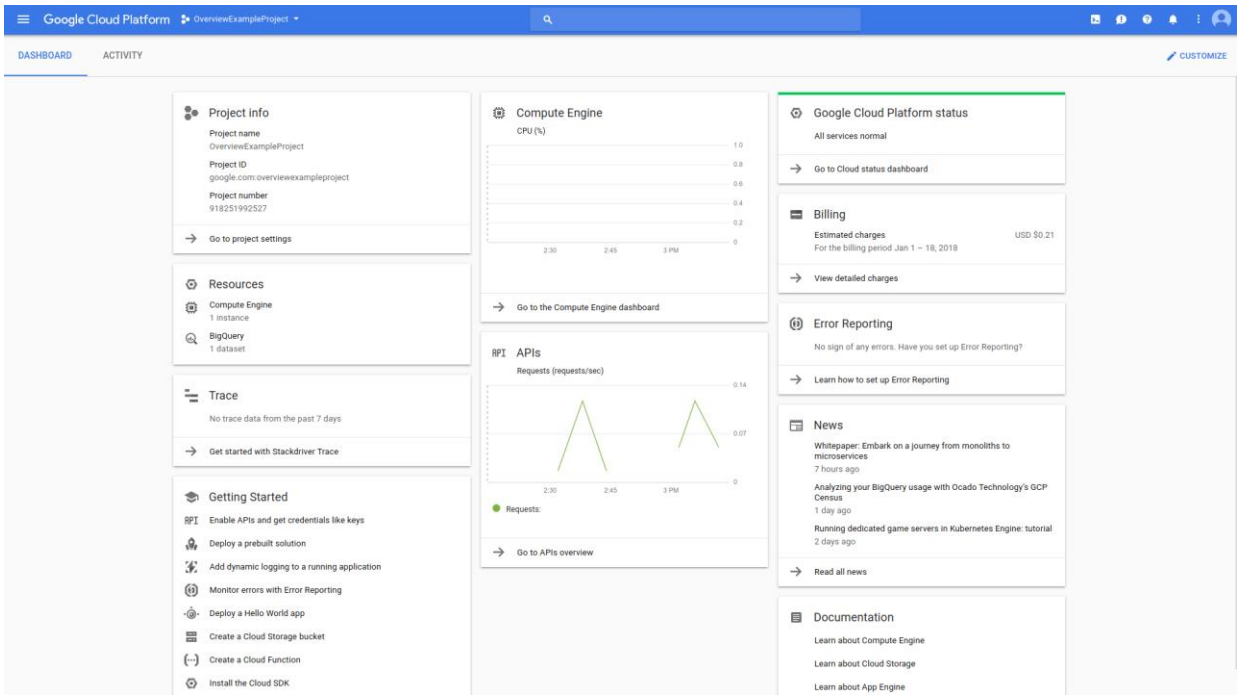

Рисунок 4.11 – UI консолі GCP Cloud Console

Ще одна особливість GCP полягає в тому, що з реєстрації облікового запису можна скористатися 1-річним безкоштовним пробним періодом і кредитом у розмірі 300 доларів США для використання більш конкретних послуг.

Серед переліку необхідних хмарних сервісів у даного провайдера відсутні сервіс електронної пошти та сервіс реєстрації і обслуговування інтернет доменів.

4.6.3 Microsoft Azure

Microsoft Azure – це хмарна платформа Microsoft. Надає можливість розробляти, запускати програми та зберігати дані на серверах, розташованих у розподілених центрах обробки даних [24].

Microsoft Azure реалізує хмарні моделі «Платформа як послуга» (PaaS) та «Інфраструктура як послуга» (IaaS). У рамках моделі «Програмне забезпечення як послуга» (SaaS) можна використовувати як сторонні служби, так і служби Microsoft. Можливості платформи Microsoft Azure забезпечують глобальну мережу розподілених центрів обробки даних Microsoft.

На додаток до базових функцій операційної системи Microsoft Azure включає виділення ресурсів за запитом для масштабування, автоматичну синхронну реплікацію даних для більшої стійкості до збоїв і збої інфраструктури для постійної доступності. Є додаткові функції, такі як обробка.

Ще одна особливість Microsoft Azure полягає в тому, що з моменту реєстрації облікового запису користувач має право на один рік безкоштовного пробного періоду та кредит у розмірі 200 доларів США для використання більш конкретних послуг.

Серед переліку необхідних хмарних сервісів у даного провайдера відсутні сервіс електронної пошти та сервіс реєстрації і обслуговування інтернет доменів.

Зовнішній вигляд UI консолі Microsoft Azure Portal зображено на рисунку 4.12.

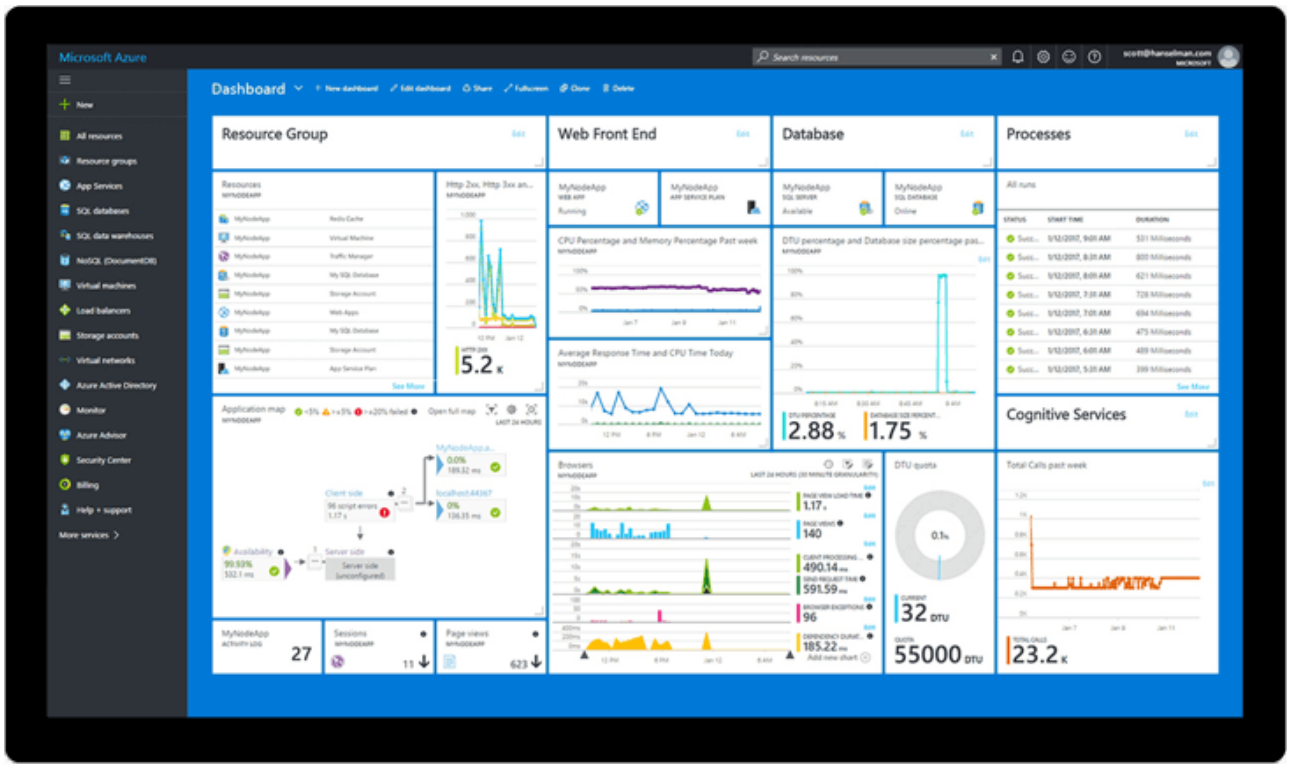

Рисунок 4.12 – UI консолі Microsoft Azure Portal

4.6.4 Порівняння цін на необхідні сервіси

Одним із ключових параметрів для аналізу хмарних постачальників є ціна за оплату використаних сервісів. На таблиці 4.1 наведені ціни за користування необхідними сервісами того або іншого хмарного провайдера.

Таблиця 4.1 – Розподіл цін за використання сервісів хмарних провайдерів

Сервіс	AWS	GCP	Azure
Хмарне сховище об’єктів, $ за 1 Gb/місяць	0.025	0.02	0.01
Бази даних, $ за 1 год	0.02	0.05	0.057
Віртуальні сервери, $ за 1 год	0.012	0.03	0.012
Балансування навантажень, $ за 1 год	0.03	0.03	0.025
Електронна пошта, $ за 1000 листів	0.10	–	–
Безсерверні обчислення, $ за мільйон запитів	0.20	0.32	0.20
Домени, $ за 1 місяць	0.50	–	–

Висновок до розділу 4

У цьому розділі на основі певних технічних та функціональних вимог було проведено аналіз стеку технологій та вибрано найсучасніші технології для розробки платформи, розробленої в цій роботі.

Сучасна мова програмування C# і платформа .NET роблять процес розробки швидким, простим та надійним. Володіючи загальною кодовою базою для всіх програм, Blazor Server стає зручним інструментом для швидкої розробки веб-додатків на вищезгаданих мовах програмування без необхідності вивчення JavaScript. EF Core спрощує та прискорює роботу з базами даних як у самій структурі, так і у вашій ERP-системі. Шаблон проектування CQRS полегшує процес розробки, розбиваючи складні служби на прості обробники запитів. База даних PostgreSQL є надійним інструментом зберігання даних фреймворку.

Так само був проведений огляд лідерів у сфері хмарних послуг з точки зору ціни на необхідні хмарні сервіси. Як можна побачити на таблиці 4.1, явного фавориту з точки зору ціни немає: деякі сервіси дешеві в одного провайдера, інші сервіси – в іншого. Наприклад, AWS переможець у сервісі баз даних, віртуальних серверів, електронної пошти, безсерверних обчислень та обслуговуванні доменів; Azure – у сервісі хмарного сховища об’єктів та балансування навантаження. GCP, з точку зору ціни на необхідні хмарні сервіси, не показав найкращого результату в жодному сервісі, а в деяких сервісах (а саме – бази даних, віртуальні сервери та безсерверні обчислення) взагалі виявився найдорожчим. Водночас такі сервіси як електронна пошта та реєстрація і обслуговування інтернет доменами у GCP та Azure відсутні взагалі. З урахуванням даного факту, а також різноманіття сервісів провайдера AWS та його позиції Квадранті інфраструктури та платформних послуг 2021 року від Gartner Research (рисунок 4.9) саме AWS було обрано в якості постачальника хмарних послуг для розробки даного фреймворку.

5 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

На основі визначених у розділі 3 функціональних та технічних вимог до системи, а також огляді технологій та шаблонів і принципів проєктування у розділі 4, було розроблено систему взаємопов’язаних додатків, кожний із яких відповідає за певну задачу.

Для зручності розробки рішення було поділене на кілька частин:

– ядро фреймворку. До даної частини належать проєкти із головними моделями, сервісами, бізнес логікою, елементами UI, програмна реалізація модулів, а також загальна логіка роботи із БД (на момент розробки була реалізована логіка для реляційних БД, в перспективі можлива реалізація також для NoSQL, графових БД тощо). На рисунку 5.1 зображені проєкти ядра фреймворку. На ньому можна побачити:

а) Aurora.ERP.Core – головний проєкт бібліотеки класів із головними моделями та сервісами фреймворку;

б) Aurora.ERP.Core.UI – бібліотека UI елементів;

в) Aurora.ERP.Core.Hub – «огортка» SignalR із власним додатковим функціоналом;

г) Aurora.ERP.Database.Core та Aurora.ERP.Database.Relational – загальна логіка взаємодії із БД в вигляді абстракцій та реалізація даної логіки для реляційних БД відповідно;

ґ) Aurora.ERP.Core.Modularity, Aurora.ERP.Core.Modules.Default та Aurora.ERP.Core.Modules.Management – логіка та основні моделі для впровадження модульності та 2 модулі: за замовчуванням (даний модуль буде встановлений у ERP системі за замовчуванням без необхідності користувачу вибирати його для встановлення) та модуль Управління (тестовий модуль) відповідно.

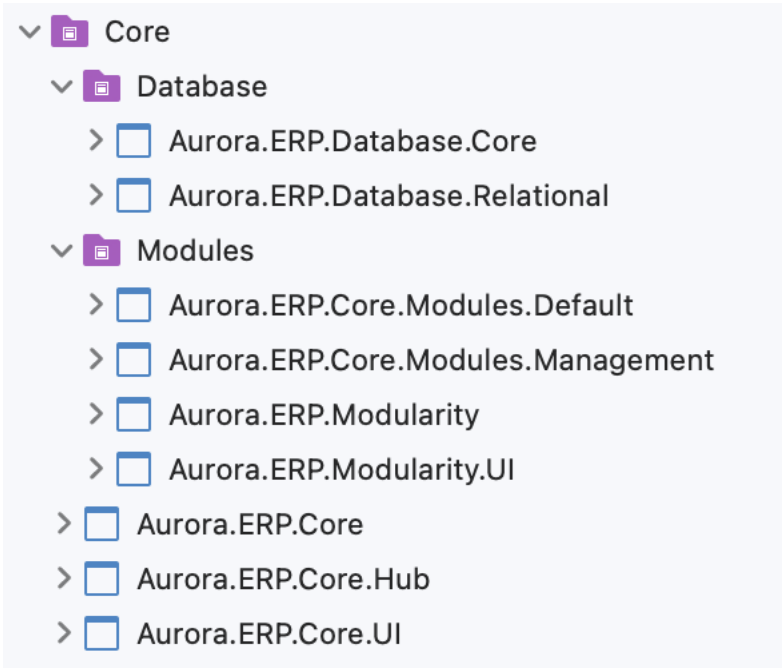

Рисунок 5.1 – Проєкти ядра фреймворку

– ERP система. До даної частини програмного рішення належать 2 проєкти: Aurora.ERP.ERPSystem.Core (головна бізнес логіка, моделі та сервіси, необхідні для коректної роботи ERP системи) та Aurora.ERP.ERPSystem.UI (власне веб-сервіс ERP системи, його UI та шлюз для обробки запитів). Дану частину можна побачити на рисунку 5.2:

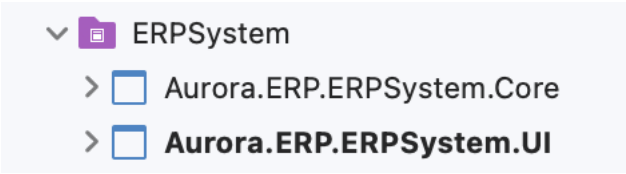

Рисунок 5.2 – Проєкти ERP системи

– інфраструктура фреймворку. До даної частини програмного рішення належать додатки для роботи із даним фреймворком для користувачів та адміністраторів фреймворку. В цій частині знаходяться проєкти головного веб-сайту, консолі взаємодії, API фреймворку та бек-офісу. Список додатків та проєктів цієї частини фреймворку зображено на рисунку 5.3.

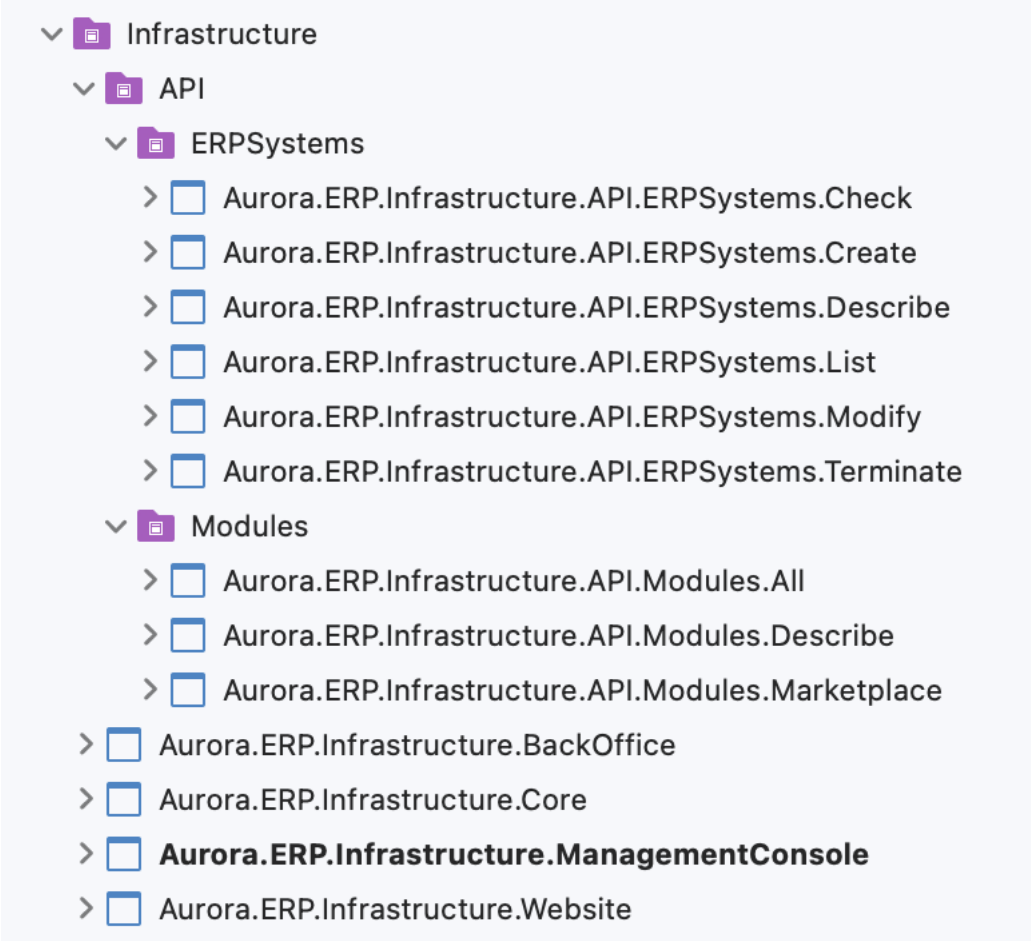

Рисунок 5.3 – Проєкти інфраструктури фреймворку

– SDK фреймворку. Однією із умов розробки було створення пакету розробки для даного фреймворку. Зазвичай IT-компанії створюють такі набори для декількох мов програмування. На даному етапі розробки такий SDK був створений лише для мови програмування C#. Список проєктів даної частини фреймворку зображено на рисунку 5.4.

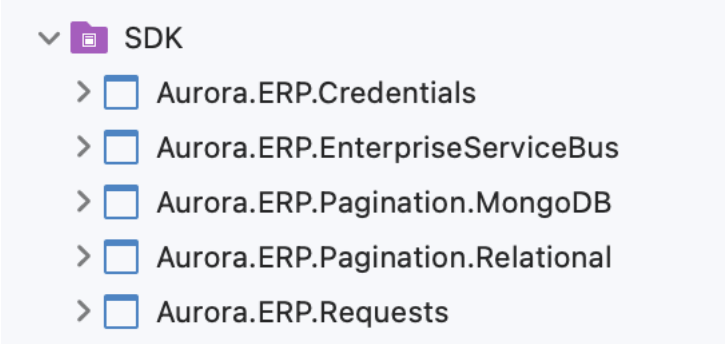

Рисунок 5.4 – Проєкти SDK фреймворку

5.1 Загальна структурна схема системи

Фреймворк, що розробляється, являє собою систему взаємопов’язаних додатків, кожний із яких відповідає за поставлену задачу, водночас працює у спільному зв’язку із іншими додатками. Загальна структурна схема фреймворку зображена на додатку А.

Як можна побачити на даній схемі, статичними додатками (додатки, що будуть опубліковані власноруч адміністраторами та власниками фреймворку) є 4 основні додатки:

– головний веб-сайт (взаємодіє повністю із усією інфраструктурною частиною фреймворку та хмарними ресурсами);

– API (взаємодіє повністю із усією інфраструктурною частиною фреймворку та хмарними ресурсами);

– бек-офіс (взаємодіє повністю із усією інфраструктурною частиною фреймворку та хмарними ресурсами, окрім можливості управління ERP системи);

– консоль управління (взаємодіє лише із створеними користувачами ERP системами).

До динамічних додатків належить лише власне веб-сервіс ERP системи – даний застосунок буде запускатися користувачами фреймворку під час запуску ERP систем. Сам застосунок (а точніше його збірка та виконуваний файл) зберігатиметься в закритому доступі у хмарі, завдяки чому жоден користувач у світі не зможе його отримати та дізнатися кодову базу додатку. Доступ до файлу матимуть лише адміністратори фреймворку.

5.2 Програмні моделі фреймворку

Для розробки даного фреймворку використовуються принципи ООП, а також для коректної роботи і зручної розробки був використаний набір моделей для абстракції реальних речей в реальному житті.

5.2.1 Моделі запитів та відповідей

Оскільки даний фреймворк, які і безліч інших систем, оснований на системі запитів-відповідей, для полегшення розробки були створені абстракції для запитів та відповідей.

Модель запиту зображена на рисунку 5.5. Дана модель складається із наступних властивостей:

– RequestID – унікальний ідентифікатор запиту. Дана властивість призначена для можливості відслідковування запитів та надає можливість реалізації своєрідного журналу запитів;

– Data – дана властивість надає можливість передавати будь-який об’єкт у тіло запиту.

```
/// <summary>
/// Base model of Aurora ERP request
/// </summary>
public abstract class AuroraERPRequest : Serializable
{
    /// <summary>
    /// Request identifier
    /// </summary>
    [JsonInclude]
    public Guid RequestID { get; set; }

    /// <summary>
    /// Request data object
    /// </summary>
    public object Data { get; set; }

    /// <summary>
    /// Initializes new instance of <see cref="AuroraERPRequest"/>
    /// </summary>
    [JsonConstructor]
    public AuroraERPRequest()
    {
        RequestID = Guid.NewGuid();
        Data = new();
    }
}
```

Рисунок 5.5 – Модель запиту у фреймворку

Модель відповіді на запит зображена на рисунку 5.6. Дана модель складається із наступних властивостей:

– ResponseID – унікальний ідентифікатор відповіді. Дана властивість призначена для можливості реалізації своєрідного журналу відповідей;

– RequestID – ідентифікатор запиту, на який була зроблена дана відповідь;

– Status – статус відповіді. У відповіді може бути 3 можливі статуси:

1) Success – відповідь матиме даний статус у випадку, якщо операція при запиті була успішна;

2) Error – відповідь матиме даний статус у випадку, коли під час обробки запиту трапилася помилка;

3) Cancelled – даний статус буде повертатися при скасуванні запиту.

– Message – повідомлення про помилку під час обробки запиту;

– Data – об’єкт із даними відповіді.

```
/// <summary>
/// Base model of Aurora ERP response
/// </summary>
public abstract class AuroraERPResponse : Serializable
{
    /// <summary>
    /// Response identifier
    /// </summary>
    [JsonInclude]
    public Guid ResponseID { get; set; }

    /// <summary>
    /// Request identifier
    /// </summary>
    public Guid RequestID { get; set; }

    /// <summary>
    /// Response status
    /// </summary>
    [ValueConverter(typeof(ResponseStatusValueConverter))]
    [JsonConverter(typeof(ResponseStatusJsonConverter))]
    public ResponseStatus Status { get; set; }

    /// <summary>
    /// Response message
    /// </summary>
    public string Message { get; set; }

    /// <summary>
    /// Response data
    /// </summary>
    public object? Data { get; set; }

    /// <summary>
    /// Initializes new instance of <see cref="AuroraERPResponse"/>
    /// </summary>
    public AuroraERPResponse()
    {
        ResponseID = Guid.NewGuid();
        RequestID = Guid.Empty;
        Message = string.Empty;
        Status = ResponseStatus.Error;
    }
}
```

Рисунок 5.6 – Модель відповіді у фреймворку

Як можна побачити на рисунках, дані моделі є абстрактними класами, і одними із успадкувань даних класів є окремі моделі запиту та відповіді у ERP системі. Ці моделі описані класами ApplicationRequest (рисунок 5.7) та ApplicationResponse (рисунок 5.8) відповідно.

```
/// <summary>
/// ERP system application request
/// </summary>
[JsonConverter(typeof(ApplicationRequestJsonConverter))]
public class ApplicationRequest : AuroraERPRequest
{
    /// <summary>
    /// Application request
    /// </summary>
    [ValueConverter(typeof(ApplicationOperationValueConverter))]
    [JsonConverter(typeof(ApplicationOperationJsonConverter))]
    [AuroraERPProperty(Required = true, ErrorMessage = "Operation is required")]
    public ApplicationOperation? Operation { get; set; }

    /// <summary>
    /// Initializes new instance of <see cref="ApplicationRequest"/>
    /// </summary>
    public ApplicationRequest()
    {
        Operation = null;
    }
}
```

Рисунок 5.7 – Модель запиту у ERP системі

```
/// <summary>
/// ERP system application response
/// </summary>
[JsonConverter(typeof(ApplicationResponseJsonConverter))]
public class ApplicationResponse : AuroraERPResponse
{
    /// <summary>
    /// Application operation
    /// </summary>
    [ValueConverter(typeof(ApplicationOperationValueConverter))]
    [JsonConverter(typeof(ApplicationOperationJsonConverter))]
    public ApplicationOperation? Operation { get; set; }

    /// <summary>
    /// Initializes new instance of <see cref="ApplicationResponse"/>
    /// </summary>
    public ApplicationResponse()
    {
        Operation = null;
    }
}
```

Рисунок 5.8 – Модель відповіді у ERP системі

Обидві моделі успадковуються від AuroraERPRequest та AuroraERPResponse відповідно та мають додаткову властивість – Operation, яка вказує, яка операція була зазначена у запиті. ApplicationOperation може набувати одного із наступних значень:

– Database operation – вказує на запит до БД. У випадку даної операції у полі Data вказується, над якою сутністю та якого модулю робиться операція;

– Module operation – вказує на конкретну операцію конкретного модуля. Кожний модуль може мати набір моделей операцій та логіку опрацювання тієї чи іншої операції;

– Add module – вказує на запит на інсталяцію нового модуля в ERP системі;

– Add module – вказує на запит на видалення модуля із ERP системи.

В залежності від значення Operation у полі Data запиту мають бути вказані конкретні дані, необхідні для обробки тієї або іншої операції.

5.2.2 Моделі фільтрації даних та розбиття на сторінки

Оскільки даний фреймворк розробляється для створення веб-сервісів ERP систем для комерційного бізнесу, бази даних такого сегменту можуть містити терабайти даних. Звісно, при вибірці даних із таблиць такого масштабу може повертатися велика кількість інформації. Спроба серіалізувати цю інформацію та передати мережею може викликати проблеми через розмір цих даних. Саме тому було прийнято за стандарт необхідність реалізації механізму фільтрації вибірки та розбиття її на сторінки (так званий пейджинг, англ: «pagination»). Саме завдяки такому механізму користувачу по всьому світі при відвідуванні, наприклад, інтернет магазину не бачать відразу всі товари (яких може бути тисячі і мільйони), а завантажують їх частково, наприклад, по 50. Це набагато зменшує пропускну здатність та навантаження на мережу і систему в цілому.

Саме в рамках цього стандарту аналогічний механізм був реалізований у даному фреймворку. Реалізований він у вигляді трьох моделей: Expression (рисунок 5.9), Filter (рисунок 5.10) та PagedResult (рисунок 5.11).

На рисунку 5.9 зображена модель логічного вираження для фільтрації вибірки даних за параметрами. Дана модель зображена на рисунку 5.10 та має дві властивості: Text (текст вибірки) та Values (значення параметрів, які вказуються при вибірці).

```
/// <summary>
/// Filtration expression model
/// </summary>
public struct Expression
{
    /// <summary>
    /// Expression text
    /// </summary>
    public string Text { get; set; }

    /// <summary>
    /// Expression values
    /// </summary>
    public IEnumerable<object> Values { get; set; }

    /// <summary>
    /// Initializes new instance of <see cref="Expression"/>
    /// </summary>
    public Expression()
    {
        Text = string.Empty;
        Values = Array.Empty<object>();
    }
}
```

Рисунок 5.9 – Модель логічного вираження для фільтрації

Формат тексту вираження аналогічний до звичайного SQL запиту для вибірки, в якому параметри беруться із колекції Values за індексом, вказаного після символу ‘@’. Наприклад, нехай існує таблиця замовлень і необхідного зробити наступну вибірку: дістати всі завершені замовлення (статус Completed) в Києві або Одесі за 1 грудня 2022 року. Тоді текст вираження буде: «((Status == @0) AND (City.Contains(@1) OR City.Contains(@2) AND (Date == @3))», тоді як значення будуть: ‘Completed’, ‘Kyiv’, ‘Odessa’ та ‘2022.12.01’. Такий формат є зрозумілим як для розробників, так і для звичайних користувачів.

На рисунку 5.10 зображена модель фільтру. Саме ця модель буде вказана при запиті і на її основі буде виконана логіка фільтрації даних. Вона складається із наступних властивостей:

– PageIndex – вказує номер сторінки під час розбиття даних на сторінки. Мінімальне значення – 1;

– PageSize – вказує максимальну кількість записів для витягнення на вказаній сторінці. За замовчуванням – 25;

– OrderBy – вказує назву колонки в таблиці, по якій треба робити сортування;

– OrderDirection – вказує тип сортування: Ascending (від меншого до більшого) або Descending (від більшого до меншого). За замовчуванням – Descending;

– Expression – модель логічного вираження;

– FilterPageLimit – вказує чи потрібно розбивати вибірку на сторінки (Limited) або ні (Unlimited). За замовчуванням – Limited.

```
/// <summary>
/// Model for filtering collections (using <c>Expression</c> property),sorting, ordering and paging
/// </summary>
public class Filter : Serializable
{
    /// <summary>
    /// Index of page to display
    /// </summary>
    public int PageIndex { get; set; }

    /// <summary>
    /// Amount of elements to display per one page
    /// </summary>
    public int PageSize { get; set; }

    /// <summary>
    /// Name of property to order by
    /// </summary>
    public string OrderBy { get; set; }

    /// <summary>
    /// Ordering direction - <c>Ascending</c> or <c>Descending</c>.
    /// Default value - <c>Descending</c>
    /// </summary>
    public string OrderDirection { get; set; }

    /// <summary>
    /// Filtration expression model
    /// </summary>
    [JsonConverter(typeof(ExpressionJsonConverter))]
    public Expression Expression { get; set; }

    /// <summary>
    /// Defines whether collection should filtered (<c>Limited</c>) or not (<c>Unlimited</c>).
    /// Default value - <c>Limited</c>
    /// </summary>
    public FilterPageLimit PageLimit { get; set; }

    [JsonConstructor]
    public Filter(FilterPageLimit pageLimit = FilterPageLimit.Limited)
    {
        PageLimit = pageLimit;

        PageIndex = GlobalConstants.DefaultPageIndex;
        PageSize = GlobalConstants.DefaultPageSize;
        OrderDirection = GlobalConstants.Descending;
        OrderBy = string.Empty;
        Expression = new();
    }
}
```

Рисунок 5.10 – Модель фільтру для вибірки даних

Останньою із трьох моделей для механізму фільтрації вибірки є власне результат даної фільтрації, який і буде повертатися у тілі відповіді на запит. Ця модель описана класом PagedResult та зображена на рисунку 5.11. Дана модель складається із наступних властивостей:

– TotalItems – вказує загальну кількість записів в даній таблиці у БД;

– PageIndex– вказує номер сторінки;

– PageSize– вказує максимальну кількість записів для витягнення;

– TotalPages– вказує загальну кількість сторінок, на яку можна поділити усі записи при зазначеній PageSize;

– FirstItemIndex– вказує індекс першого запису в рамках усієї таблиці;

– LastItemIndex– вказує індекс останнього запису в рамках усієї таблиці;

– List – список отриманих записів на поточній сторінці після фільтрації та сортування;

– HasPreviousPage – вказує чи знаходиться даний результат на першій сторінці вибірки;

– HasNextPage – вказує чи знаходиться даний результат на останній сторінці вибірки.

Дана модель є своєрідною останньою ланкою у ланцюзі обробки фільтрації вибірки. Спочатку створюється об’єкт логічного вираження, в якому вказуються параметри вибірки (які поля фільтрувати та за якими умовами), далі відбувається ініціалізація об’єкту фільтру із параметрами сортування, розбиття на сторінки та вираженням. Далі даний фільтр передається в якості даних запитом, передається до обробника взаємодії із БД (даних механізм використовується як у веб-сервісі ERP системи, так і у додатках інфраструктури фреймворку), і після обробки у тілі відповіді повертається об’єкт класу PagedResult із результатами фільтрації та вибірки.

```
public class PagedResult : Serializable
{
    /// <summary>
    /// Total amount of items in source <b>before</b> filtering
    /// </summary>
    public long TotalItems { get; set; }

    /// <summary>
    /// Index of current page. Default value - 1
    /// </summary>
    public int PageIndex { get; set; }

    /// <summary>
    /// Amount of elements to display per one page
    /// </summary>
    public int PageSize { get; set; }

    /// <summary>
    /// Total amount of pages. Calculated as <c>TotalItems</c> / <c>PageSize</c>
    /// </summary>
    [JsonInclude]
    public int TotalPages { get; set; }

    /// <summary>
    /// Index of first item of current page`s collection within the query. Default value is 1, if collection is empty - 0
    /// </summary>
    [JsonInclude]
    public long FirstItemIndex { get; set; }

    /// <summary>
    /// Index of last item of current page`s collection within the query
    /// </summary>
    [JsonInclude]
    public long LastItemIndex { get; set; }

    /// <summary>
    /// Collection of items
    /// </summary>
    [JsonIgnore(Condition = JsonIgnoreCondition.WhenWritingNull)]
    public IList? List { get; set; }

    /// <summary>
    /// Defines whether current page index is more than 1
    /// </summary>
    public bool HasPreviousPage => PageIndex > GlobalConstants.DefaultPageIndex;

    /// <summary>
    /// Defines whether current page index is less than total amount of pages
    /// </summary>
    public bool HasNextPage => PageIndex < TotalPages;
```

Рисунок 5.11 – Модель результату фільтрації вибірки

5.2.3 Модель унікальної назви ресурсу (ERN)

Концепція унікального ідентифікатору ресурсу була розроблена компанією Amazon. Оскільки дана IT-компанія є постачальником безлічі IT сервісів в різних регіонах світу, постало питання ідентифікації ресурсу, до якого користувач намагається отримати доступ. Механізм цієї ідентифікації отримав назву Amazon Resource Name (ARN) [25]. ARN – це формат найменування ресурсів, який використовується для ідентифікації певного ресурсу в публічній хмарі Amazon Web Services (AWS). ARN, які є специфічними для AWS, допомагають адміністратору відстежувати та використовувати елементи й політики AWS у продуктах AWS і

викликах API. Окремі елементи синтаксису ARN називаються просторами імен, і користувачі повинні ввести відповідний простір імен, щоб створити функціональний ARN. Розділ і служба, перша та друга частини угоди про іменування, відповідають відповідним розташуванням, у яких знаходиться ресурс. Ідентифікатор облікового запису означає номер облікового запису користувача AWS, який може бути потрібним або не потрібним залежно від ресурсу. Загальний форматом ARN є: «arn:partition:service:region:account-id:resource-id».

В даному фреймворку був розроблений аналог даного механізму та отримав назву Entity Resource Name (ERN). Розроблений він був із метою ідентифікації, до якого модуля відноситься той чи інший ресурс. Даний функціонал має наступний синтаксис: «ern:module-prefix:entity:entity-id», та складається із наступних частин:

– module-prefix – префікс модуля (вказується у моделі модуля в якості атрибуту). Вказує, до якого модуля належить ресурс;

– entity – префікс сутності модуля (вказується у моделі сутності в якості атрибуту). Вказує, яку сутність модулю слід використати;

– entity-id – ідентифікатор вказаної сутності. Дана частина може мати 2 значення:

1) * (wildcard-запис) – вказує на загальну вибірку із таблиці сутності;

2) ########-####-####-####-############ – ідентифікатор конкретного елемента таблиці у вигляді GUID.

Для зручності розробки даний механізм був аналогічно розміщений у програмній моделі, яка зображена на рисунку 5.12.

```
public struct Ern
{
    /// <summary>
    /// Module prefix
    /// </summary>
    public readonly string Module;

    /// <summary>
    /// Entity prefix
    /// </summary>
    public readonly string Entity;

    /// <summary>
    /// Entity ID or "*"
    /// </summary>
    public readonly string EntityID;

    /// <summary>
    /// Initializes an empty <see cref="Ern"/> model
    /// </summary>
    public Ern() : this(string.Empty, string.Empty, string.Empty) { }

    /// <summary>
    /// Initializes <see cref="Ern"/> model for <c>all</c> entities of <c>default</c> module
    /// </summary>
    /// <param name="entity">Entity prefix</param>
    public Ern(string entity) : this(GlobalConstants.DefaultModulePrefix, entity) { }

    /// <summary>
    /// Initializes <see cref="Ern"/> model for <c>all</c> entities of specified module
    /// </summary>
    /// <param name="module">Module prefix</param>
    /// <param name="entity">Entity prefix</param>
    public Ern(string module, string entity) : this(module, entity, GlobalConstants.Wildcard) { }

    /// <summary>
    /// Initializes <see cref="Ern"/> model for entity with specified ID of specified module
    /// </summary>
    /// <param name="module">Module prefix</param>
    /// <param name="entity">Entity prefix</param>
    /// <param name="entityID">Entity ID</param>
    public Ern(string module, string entity, string entityID)
    {
        Module = module;
        Entity = entity;
        EntityID = entityID;
    }
}
```

Рисунок 5.12 – Програмна модель ERN

Як можна побачити на малюнку, дана модель має 3 варіанти ініціалізації:

– тільки для сутності (в такому випадку формується ERN для загальної вибірки вказаної сутності із модуля за замовчуванням);

– модуль та сутність (в такому випадку формується ERN для загальної вибірки вказаної сутності із вказаного модуля);

– модуль, сутність та ідентифікатор (в такому випадку формується ERN для конкретної вказаної сутності із вказаного модуля).

5.3 Приклад програмної реалізації модуля

Як вже було зазначено раніше, концепція модульності дозволить розділити бізнес-логіку на окремі компоненти. З точки зору розробки, дана концепція також стане в нагоді, оскільки в такому випадку розробка, тестування та впровадження кожного модуля відбуватиметься окремо. Завдяки цьому розробкою кожного модуля кожна команда розробників займатиметься окремо. Звідси маємо одночасну розробку багатьох модулів одночасно, що пришвидшить їхнє впровадження на ринку та забезпечить різноманіття вибору необхідного функціоналу для користувачів фреймворку.

Проєкт модуля являтиме собою бібліотеку класів Razor – звичайна бібліотека класів, з єдиною відмінністю в тому, що надає можливість створювати Razor компоненти – основними UI елементами для Blazor додатків. Приклад такого проєкту зображено на рисунку 5.13.

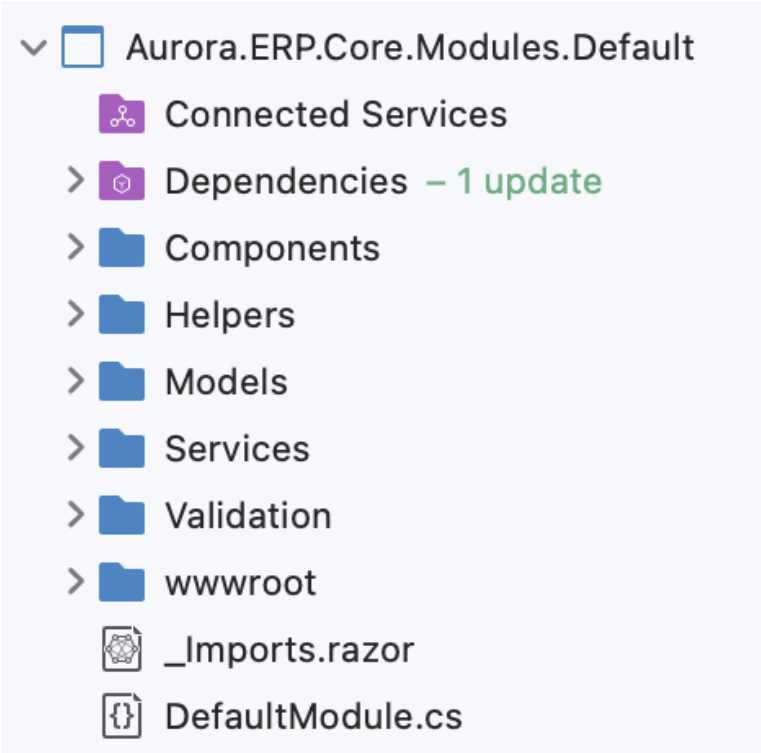

Рисунок 5.13 – Приклад бібліотеки модуля

Як можна побачити, структура проєкту складається із наступних папок:

– Components: саме у цій папці зберігатимуться вищезгадані Razor компоненти – таблиці даних, форми для заповнення, ті або інші візуальні елементи, через які користувачі будуть взаємодіяти із ERP системою;

– Models: сутності даного модуля. Саме ці моделі будуть використовуватись для бізнес-логіки даного модуля;

– Services: абстракції та реалізації сервісів даного модуля;

– Validation: папка із логікою валідації тих або інших сутності при їхньому створенні;

– wwwroot: папка із статичним контентом, таким як файли стилів, зображення, веб-скрипти тощо.

Кожний модуль модель із трьома властивостями:

– Name: назва модуля, наприклад «Project management» або «Logistic», може бути локалізована для того чи іншого ринку;

– Prefix: префікс модуля, наприклад «project-management» або «logistic»;

– Type: тип модуля, назва його класу, на рисунку 5.13 це DefaultModule.

```
[PluralTitle("Modules")]
public class Module : SerializableEntity
{
    [Filter(nameof(GlobalConstants.Search))]
    [Display(Name = "Name")]
    public string Name { get; set; }

    [Filter(nameof(GlobalConstants.Search))]
    [Display(Name = "Prefix")]
    public string Prefix { get; set; }

    [Filter(nameof(GlobalConstants.Search))]
    [Display(Name = "Type")]
    public string Type { get; set; }

    public Module() : this(string.Empty, string.Empty, string.Empty) { }

    [JsonConstructor]
    protected Module(string name, string prefix, string type)
    {
        Name = name;
        Prefix = prefix;
        Type = type;
    }
}
```

Рисунок 5.14 – Модель модуля

5.4 Реалізація ERP безпеки через механізм HMAC SHA256

Однією із основних технічних вимог було створення механізму забезпечення конфіденційності та надійності обробки запитів. В якості даного механізму був вибраний алгоритм HMAC SHA256.

HMAC (Hash-based Message Authentication Code) – це механізм перевірки цілісності інформації, що передається або зберігається в ненадійних середовищах [26]. Такі методи є невід'ємною та необхідною частиною світу відкритих обчислень та комунікацій. Механізми, які забезпечують такі перевірки цілісності на основі закритого ключа, зазвичай називають кодами автентифікації повідомлень (MAC). MAC зазвичай використовуються між двома сторонами, які спільно використовують секретний ключ для перевірки автентичності інформації, що надсилається між цими сторонами. Цей стандарт визначає MAC. Механізм, що використовує криптографічну хеш-функцію у поєднанні із закритим ключем, називається HMAC. Перевагами HMAC є наступні фактори:

– можливість використання хеш-функцій, які вже є в програмному продукті;

– відсутність необхідності внесення змін до реалізації існуючих хеш-функцій (внесення змін може призвести до погіршення продуктивності та погіршення криптостійкості);

– можливість заміни хеш-функції у разі появи більш безпечної або швидшої хеш-функції.

На додатку Б зображена блок-схема алгоритму підпису запиту. Загалом він складається із чотирьох кроків (рисунок 5.15), кожний із яких має більш детальну і складну логіку:

1) створити канонічний запит;

2) використати канонічний запит і додаткові метадані, щоб створити рядок для підпису;

3) отримайте ключ підпису зі свого секретного ключа доступу. Потім використати ключ підпису та рядок із кроку №3, щоб створити підпис;

4) додати отриманий підпис до HTTP-запиту в вигляді заголовку.

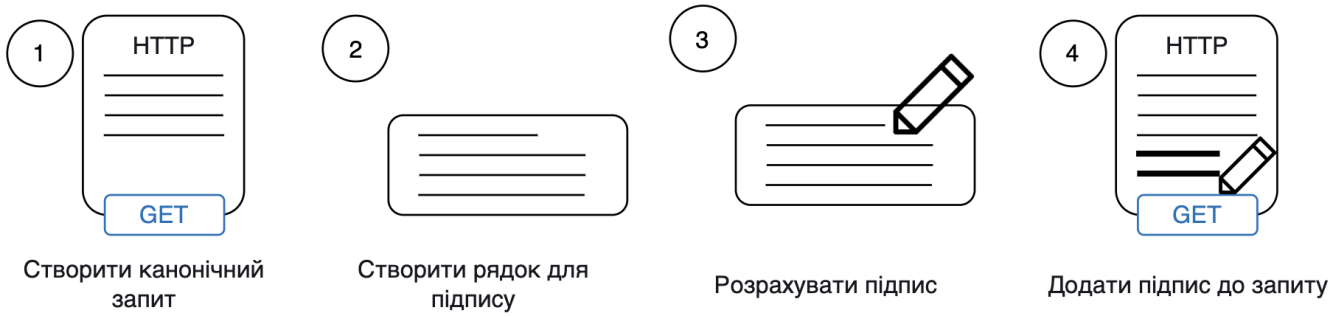

Рисунок 5.15 – Загальні кроки алгоритму підпису запиту

5.4.1 Створення канонічного запиту

Канонічний запис запиту – запис, що обраховується за формулою 5.1 [27]:

$$CanonicalRequest = HTTPRequestMethod + '\n' + CanonicalURI + '\n' + CanonicalQueryString + '\n' + CanonicalHeaders + '\n' + SignedHeaders + '\n' + HexEncode(Hash(RequestPayload)) \quad (5.1)$$

У цій формулі *Hash* представляє функцію, яка створює дайджест повідомлення, зазвичай SHA-256. *HexEncode* представляє функцію, яка зміст у hex форматі дайджесту символами нижнього регістру. Кожен вхідний байт повинен бути представлений рівно двома шістнадцятковими символами.

Щоб створити канонічний запит, потрібно об’єднати наступні компоненти з кожного кроку в єдиний рядок:

1) починається з методу запиту HTTP (GET, PUT, POST тощо) та символ нового рядку ‘\n’;

2) додати канонічний шлях запиту з подальшим символом ‘\n’. Канонічний шлях – це закодована версія компонента абсолютного шляху (від хоста HTTP до символу початку параметрів "?"). Якщо абсолютний шлях порожній, потрібно використати символ /;

3) додати канонічний рядок запиту з подальшим символом ‘\n’. Якщо запит не містить рядка запиту, використати порожній рядок. Для побудови канонічного рядка запиту, потрібно виконати наступні кроки:

а) відсортувати імена параметрів за у порядку зростання. Закодувати у форматі URI кожне ім'я та значення параметрів. Створити рядок канонічного запиту, починаючи з першого імені параметра у списку відсортованих;

б) для кожного параметра додати URI-кодовану назву параметра з подальшим символом = та подальшим URI-кодованим значенням параметра. Використати порожній рядок для параметрів, які не мають значення;

в) додати символ & після кожного значення параметра, за винятком останнього значення у списку.

4) додати канонічні заголовки, за якими слідує ‘\n’. Канонічні заголовки складаються зі списку всіх заголовків HTTP, які будуть включені із підписаним запитом. Щоб створити список канонічних заголовків, потрібно конвертувати всі назви заголовків у нижній регістр.

Канонічний список заголовків будується за формулою 5.2:

$$CanonicalHeaders = CanonicalHeadersEntry0 + CanonicalHeadersEntry1 + ... + CanonicalHeadersEntryN, \quad (5.2)$$

де $CanonicalHeadersEntry = Lowercase(HeaderName) + ':' + Trimall(HeaderValue) + '\n'$.

Lowercase являє собою функцію, яка перетворює всі символи в малі літери. Функція Trimall видаляє зайвий пробіл до та після значень і перетворює послідовні пробіли в один пробіл. Створити список канонічних заголовків потрібно сортуючи отримані заголовки за допомогою коду символів, а потім ітеруючи через назви заголовків.

5) додати підписані заголовки, а потім ‘\n’. Це значення – це список заголовків, які були включені в канонічні заголовки. Таким чином вказується, які заголовки у запиті є частиною процесу підписання, а які можуть бути проігноровані для перевірки

запиту. Щоб створити список підписаних заголовків, необхідно конвертувати всі імена заголовків у нижній регістр, відсортувати їх та відокремити крапкою з комою. Список підписаних заголовків обраховується за формулою 5.3:

$$SignedHeaders = Lowercase(HeaderName0) + ';' + Lowercase(HeaderName1) + ";" + ... + Lowercase(HeaderNameN), \quad (5.3)$$

6) використати хеш-функцію (дайджест), наприклад SHA256, щоб створити хешоване значення тіла запиту HTTP або HTTPS. Обчислення захешованого тіла запиту відбувається за формулою 5.4:

$$HashedPayload = Lowercase(HexEncode(Hash(requestPayload))), \quad (5.4)$$

Під час створення рядку для підпису, необхідно вказати алгоритм підпису, який використовувався для хешування корисного навантаження. В даному випадку це HMAC-SHA256. Хешоване тіло запиту має бути представлене як шістнадцятковий рядок у нижньому регістрі. Якщо тіло запиту порожнє, потрібно використати порожній рядок як вхідні дані для хеш-функції.

7) щоб створити готовий канонічний запит, потрібно об’єднати всі компоненти з кожного кроку в один рядок. Як зазначалося, кожен компонент закінчується символом нового рядка;

8) створити хеш канонічного запиту за тим самим алгоритмом, який був використаний для хешування тіла запиту. Хешований канонічний запит має бути представлений у вигляді рядка шістнадцяткових символів нижнього регістру.

Отриманий хеш буде використаний як частина рядка для підпису.

5.4.2 Створення рядка для підпису

Рядок для підпису містить метаінформацію про запит і про канонічний запит. Саме цей рядок для підпису та похідний його ключ підпису використовуються як вхідні дані для обчислення підпису. Щоб створити рядок для підпису, потрібно об’єднати алгоритм і хеш канонічного запиту, як показано в формулі 5.5:

$$StringToSign = Algorithm + \backslash n + HashedCanonicalRequest \quad (5.5)$$

В результаті маємо 2 рядки: в першому вказаний HMAC-SHA256 (алгоритм, що використовувався), а в другому – хеш канонічного запиту.

5.4.3 Розрахунок підпису

Обчислення підпису відбувається за допомогою секретного ключа доступу.

Щоб обрахувати підпис, потрібно виконати наступні кроки:

1) отримати ключ підпису. Для цього потрібно скористатися секретним ключем доступу, щоб створити серію кодів автентифікації повідомлень на основі хеш-кодів (HMAC). Це показано в формулі 5.6:

$$SigningKey = HMAC(StringToSign, SecretAccessKey), \quad (5.6)$$

де *HMAC(key, data)* представляє функцію HMAC-SHA256, яка повертає вихідні дані у двійковому форматі.

2) обчислити підпис. Для цього потрібно використати ключ підпису, який був отриманий, і рядок для підпису як вхідні дані для ключової хеш-функції. Після обчислення підпису необхідно перетворити двійкове значення в шістнадцяткове. Обчислення підпису відбувається за формулою 5.7:

$$Signature = HexEncode(HMAC(SigningKey, StringToSign)) \quad (5.7)$$

5.4.4 Приєднання підпису до запису

Після того обчислення підпису, його необхідно додати до запиту. У даному фреймворку це відбувається шляхом приєднання підпису до заголовка HTTP запиту під назвою Authorization. Вміст заголовка створюється після обчислення підпису, як описано в попередніх кроках, тому заголовок авторизації не входить до списку підписаних заголовків. Хоча заголовок називається Authorization, інформація про підпис фактично використовується для автентифікації.

Контент заголовку Authorization записується у наступному форматі (формула 5.8):

Authorization*: Algorithm* Credential=*AccessKey*, Signature=*Signature*, (5.8)

де *Algorithm* – назва алгоритму, що використовувався (в даному випадку – HMAC-SHA256), *AccessKey* – публічний ключ (використовується для перевірки підпису), *Signature* – отриманий раніше підпис запиту.

Коли API фреймворку або ERP системи отримує запит, вона виконує ці самі дії, щоб обчислити підпис, який був надісланий у запиті. Потім вона порівнює свій розрахований підпис із тим, який був надісланий надіслали разом із запитом. Якщо підписи збігаються, запит обробляється. Якщо підписи не збігаються, запит відхиляється.

5.5 Використані хмарні сервіси

Як було зазначено у розділі 4, головним постачальником хмарних послуг був обраний IT гігант AWS, завдяки своєму різноманіттю сервісів та доступним цінам. При розробці даного фреймворку був використаний цілий ряд сервісів AWS.

5.5.1 Сервіс AWS S3

Amazon Simple Storage Service (Amazon S3) – це сервіс зберігання об’єктів, який пропонує найкращі в галузі масштабованість, доступність даних, безпеку та продуктивність. Даний сервіс використаний в якості зберігання не тільки статичного контенту, але і динамічного. До динамічного контенту належать файли бібліотек нових модулів, які потім будуть вивантажуватись при установці модуля до ERP системи; звіти по статистиці по користувачам фреймворку тощо. Саме тут зберігатиметься і періодично оновлюватиметься виконуваний файл веб-сервісу ERP системи. До статичного контенту належать статичні зображення та HTML документи-шаблони для електронних повідомлень. В перспективі даний сервіс може бути використаний для хостингу статичного веб-сайту документації даного фреймворку.

Дані зберігаються як об'єктів у ресурсах, званих «сегментами», і обсяг одного об'єкта може досягати 5 терабайт. Функції S3 включають додавання тегів метаданих до об'єктів, переміщення та зберігання даних між класами сховища S3, налаштування та застосування засобів керування доступом до даних, захист даних від неавторизованих користувачів, виконання аналізу великих даних. Включає такі функції, як відстеження та перегляд даних на рівні об'єктів та сегментів, використання сховища та тенденції активності у організації. Доступ до об'єктів можна отримати через точку доступу S3 або через ім'я хоста кошика.

Amazon S3 забезпечує найкращу в галузі продуктивність хмарного сховища об'єктів. Amazon S3 підтримує паралельні запити. Це дозволяє масштабувати продуктивність S3 за допомогою обчислювального кластера без налаштування програми. Продуктивність Amazon S3 підтримує не менше 3500 запитів на секунду на додавання даних і 5500 запитів на секунду на вилучення даних. Кожен префікс S3 може підтримувати швидкість запитів, що значно підвищує продуктивність.

Для зберігання інформації по даному фреймворку був створений відповідний бакет, зміст якого зображено на рисунку 5.16. Як можна побачити, даний сервіс дозволяє створювати папки, по яким можна зручно розподіляти файли. На даний момент створені наступні папки:

– assets: у даній папці зберігатимуться статичні файли. Наразі там знаходяться вже згадані HTML шаблони для електронних листів;

– billing-reports: оскільки даний фреймворк в перспективі може стати комерційним, важливо розробити механізм ведення статистики та відслідковувати оплату користувачів за надані послуги. Звіти по податковим періодам та інша фінансова інформація зберігатиметься саме у цій папці;

– erp-system: саме у цій папці знаходитиметься виконуваний файл із веб-сервісом ERP системи. Дана папка є приватною, тому доступ до вказаного файлу можливий лише із приватної мережі самого AWS, що гарантує збереженість інформації про реалізацію веб-сервісу на рівні коду;

– modules: структурно ця папка поділятиметься на під-папки із назвами префіксів відповідних модулів (на рисунку 5.17 можна побачити приклад даної ієрархії для тестового модуля Management). У папці відповідного модуля зберігатиметься програмний файл модуля, а також будь-яка інша корисна інформація.

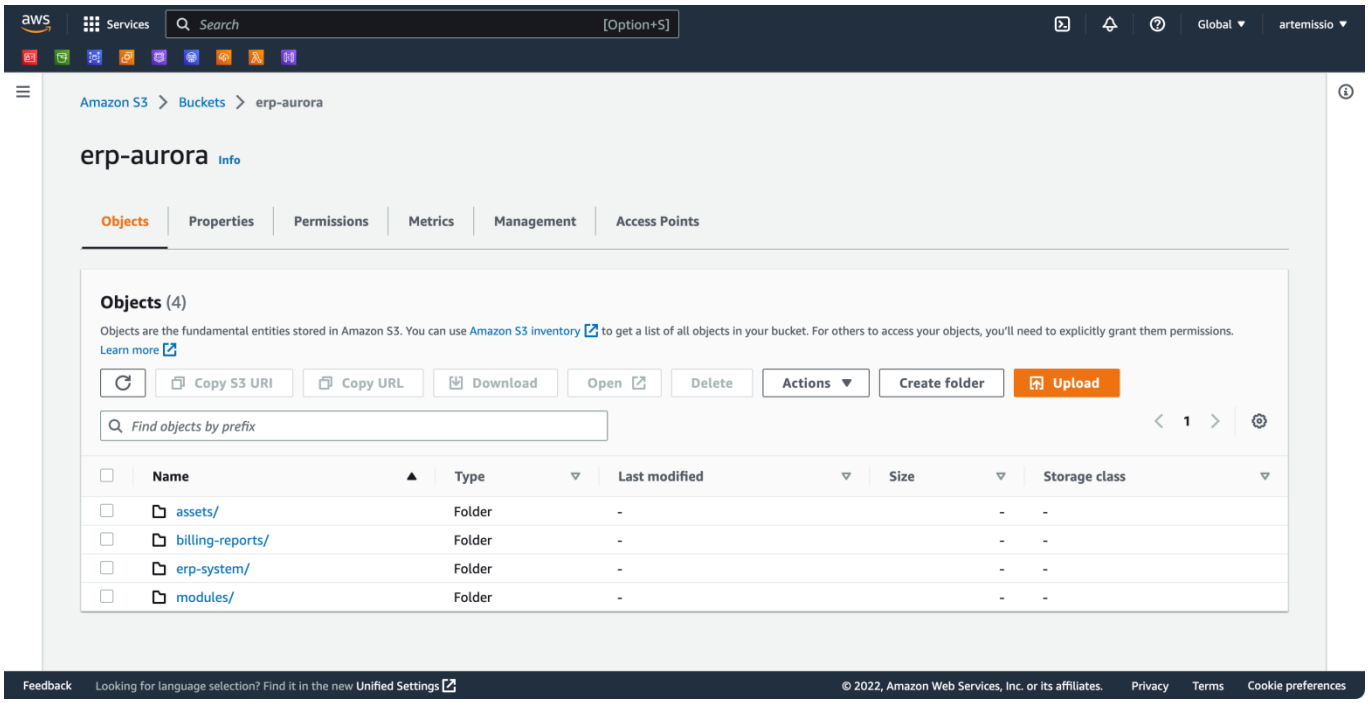

Рисунок 5.16 – Зміст S3 бакету для даного фреймворку

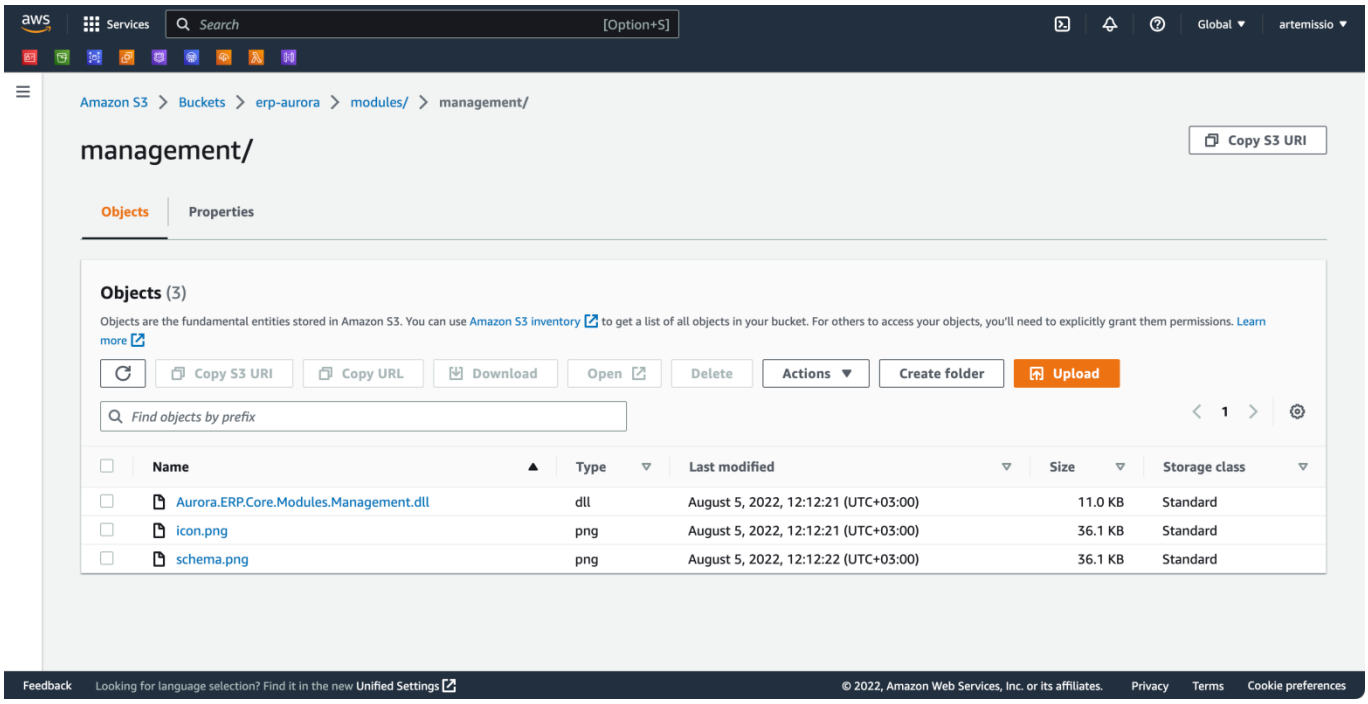

Рисунок 5.17 – Зміст папки модуля в S3 бакеті

5.5.2 Сервіс AWS EC2

Amazon Elastic Compute Cloud (Amazon EC2) – це сервіс AWS, який дозволяє користувачам орендувати віртуальні машини, які називаються екземплярами. Запустіть інстанси за допомогою попередньо налаштованих образів (Amazon Machine Images – AMI), щоб скоротити час завантаження нових серверів. Взаємодіяти зі службою можна за допомогою веб-інтерфейсу, інтерфейсу командного рядка та програмно за допомогою API.

З Amazon EC2 можна швидше розробляти та розгортати програми, оскільки не потрібно вкладати кошти в обладнання. Використовуючи Amazon EC2, користувачі можуть запускати будь-яку кількість віртуальних серверів, налаштовувати безпеку та мережу, а також керувати сховищем. Amazon EC2 знижує потребу в прогнозуванні трафіку, дозволяючи збільшувати або зменшувати масштаб у міру зміни попиту.

Amazon EC2 надає можливість розміщувати екземпляри у кількох місцях. Розташування Amazon EC2 складається з регіонів та зон доступності. Зона доступності – це окреме розташування, ізольоване від збоїв в інших зонах доступності

та призначене для забезпечення недорогого підключення до мережі з малою затримкою до інших зон доступності в тому ж регіоні. Запускаючи екземпляри в окремих зонах доступності, користувачі можуть запобігти збою своїх програм в одному місці. Регіони складаються з однієї або кількох зон доступності та географічно розподілені.

Amazon EC2 надає такі функції:

– віртуальне обчислювальне середовище, зване екземпляром;

– попередньо налаштовані шаблони для екземплярів, які називаються образами машин Amazon (AMI). Упакуйте дані, необхідні серверу, включаючи операційну систему та додаткове програмне забезпечення;

– різні конфігурації ЦП, пам'яті та пропускної спроможності мережі для екземплярів;

– захист інформації для входу в екземпляри за допомогою пар ключів (AWS зберігає відкритий ключ, а зберігайте закритий ключ у безпечному місці);

– том сховища для тимчасових даних, який видаляється, коли користувач зупиняє, переводить у сплячий режим або видаляє екземпляр, який називається томом сховища екземплярів;

– об'єми постійного сховища даних із використанням Amazon Elastic Block Store (Amazon EBS), які називаються томами;

– кілька фізичних місць для ресурсів, таких як екземпляри та томи Amazon EBS, які називаються регіонами та зонами доступності;

– брандмауер, що дозволяє вказати протоколи, порти та діапазони вихідних IP-адрес, які можуть обмежувати доступ до ваших екземплярів;

– статична IPv4-адреса для динамічних хмарних обчислень, яка називається еластичною IP-адресою;

– метадані, які називають тегами. Можна створювати та призначати ресурси Amazon EC2;

– віртуальна мережа, яка логічно ізольована від решти хмари AWS і може бути приєднана до власної мережі, яка називається віртуальною приватною хмарою (VPC).

Даний сервіс є одним із основних хмарних сервісів, оскільки він буде використовуватись найчастіше. Саме в ньому будуть опубліковані інфраструктурні додатки (веб-сайт, бек-офіс, консоль управлінні), а також будуть запускатися сервери для веб-сервісів ERP систем.

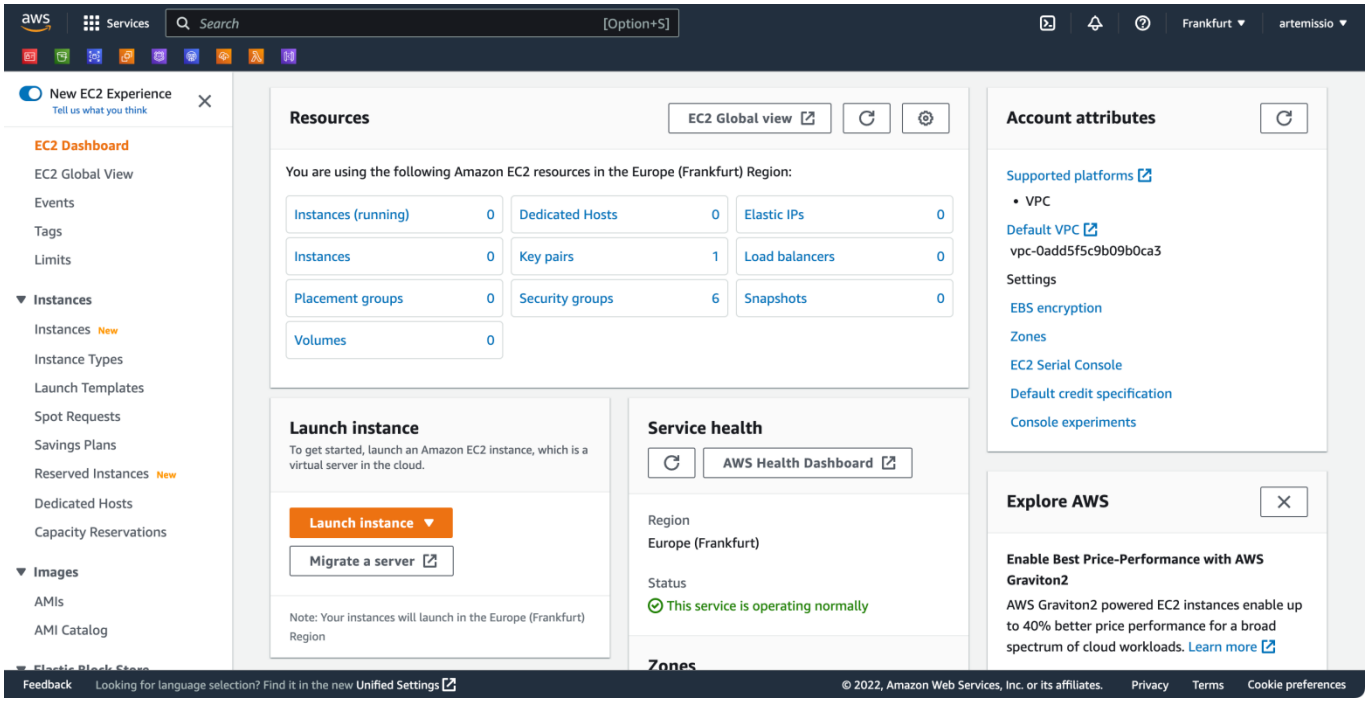

Рисунок 5.18 – UI консолі управління екземплярами сервісу EC2

5.5.3 Сервіс AWS RDS

Служба реляційних баз даних Amazon (Amazon RDS) – це набір керованих служб, які спрощують налаштування, роботу та масштабування баз даних у хмарі. Наразі існує підтримка 7 провайдерів БД – Amazon Aurora із сумісністю з MySQL, Amazon Aurora із сумісністю з PostgreSQL, MySQL, MariaDB, PostgreSQL, Oracle та SQL Server.

Amazon RDS спрощує використання реплікації для підвищення доступності та надійності робочих навантажень. Варіанти розгортання в кількох зонах доступності дозволяють запускати критично важливі робочі навантаження з високою доступністю та вбудованою функцією автоматичного аварійного перемикання з первинних баз

даних на синхронно репліковані вторинні бази даних. Репліки читання дозволяють масштабувати ємність за межі розгортання однієї бази даних робочих навантажень бази даних із великим обсягом операцій читання. Як і у випадку з усіма сервісами AWS, попередні інвестиції не потрібні, і користувачі платять лише за ресурси, що використовуються.

Даний сервіс використовується в якості постачання БД як для інфраструктурних додатків, так і для динамічного створення БД при створенні ERP систем. Важливо відмітити, що даний сервіс використовуватиметься у випадку SQL типу БД у вказаній ERP системі. В перспективі буде розроблена підтримка також і NoSQL, Graph баз даних тощо.

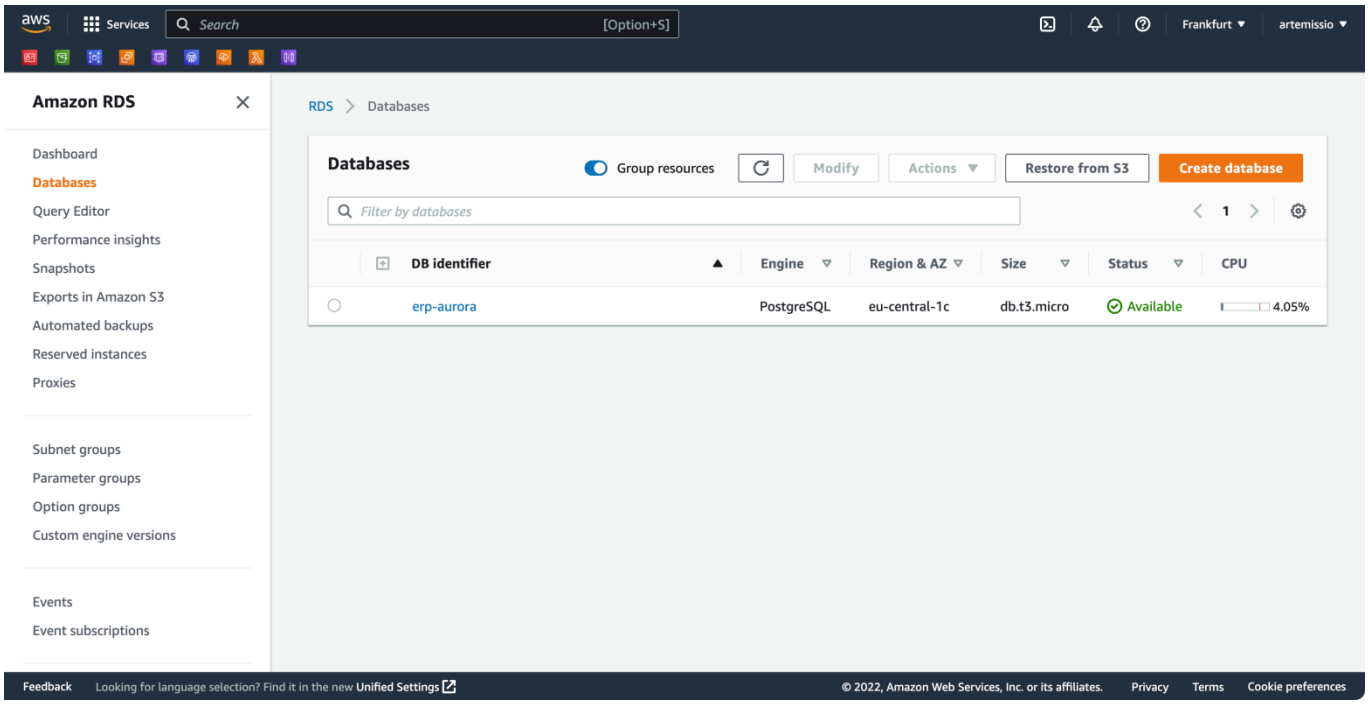

Рисунок 5.19 – UI консолі управління екземплярами БД сервісу RDS

5.5.4 Сервіс AWS ELB

Сервіс балансування навантаження (англ. «Elastic Load Balancing») автоматично розподіляє вхідний трафік програми між усіма запущеними екземплярами EC2. Еластичний баланс навантаження допомагає керувати вхідними

запитами шляхом оптимальної маршрутизації трафіку, щоб жоден екземпляр не перевантажувався.

ELB підтримує такі балансувальники навантаження:

– програм (Application Load Balancer): даний балансувальник стане в нагоді, якщо потрібен гнучкий набір функцій для програм із трафіком HTTP і HTTPS. Працюючи на рівні запиту, балансувальники навантаження додатків забезпечують розширені функції маршрутизації та видимості, націлені на архітектури додатків, включаючи мікросервіси та контейнери;

– мережевого навантаження (Network Load Balancer): даний балансувальник стане в нагоді, коли потрібна надвисока продуктивність, масштабне розвантаження TLS, централізоване розгортання сертифікатів, підтримка UDP і статичні IP-адреси для програм. Працюючи на рівні підключення, балансувальники мережевого навантаження здатні безпечно обробляти мільйони запитів на секунду, зберігаючи наднизькі затримки;

– шлюзу (Gateway Load Balancer): даний балансувальник стане в нагоді, коли потрібно розгорнути цілий стек віртуальних пристроїв сторонніх розробників, які підтримують GENEVE, і керувати ними. Ці пристрої дають змогу покращити безпеку, відповідність і правила керування;

– класичні балансувальники навантаження (Classic Load Balancer): перенаправляє трафік до конкретно вибраних EC2 екземплярів.

Даний сервіс буде використовуватися в парі із сервісом EC2 – при створенні ERP системи буде буде також створюватись балансувальник навантаження, на який і буде перенаправлятися трафік із глобального домену. Більш того, оскільки даний фреймворк в перспективі може стати комерційним, і кожним створюваним веб-сервісам ERP системи будуть користуватися багато користувачів, наявність даного балансувальника дозволить робити логічне об’єднання декількох серверів і розподіляти навантаження між ними в залежності від потреб.

На рисунку 5.20 зображена консоль управління балансувальниками навантажень сервісу ELB.

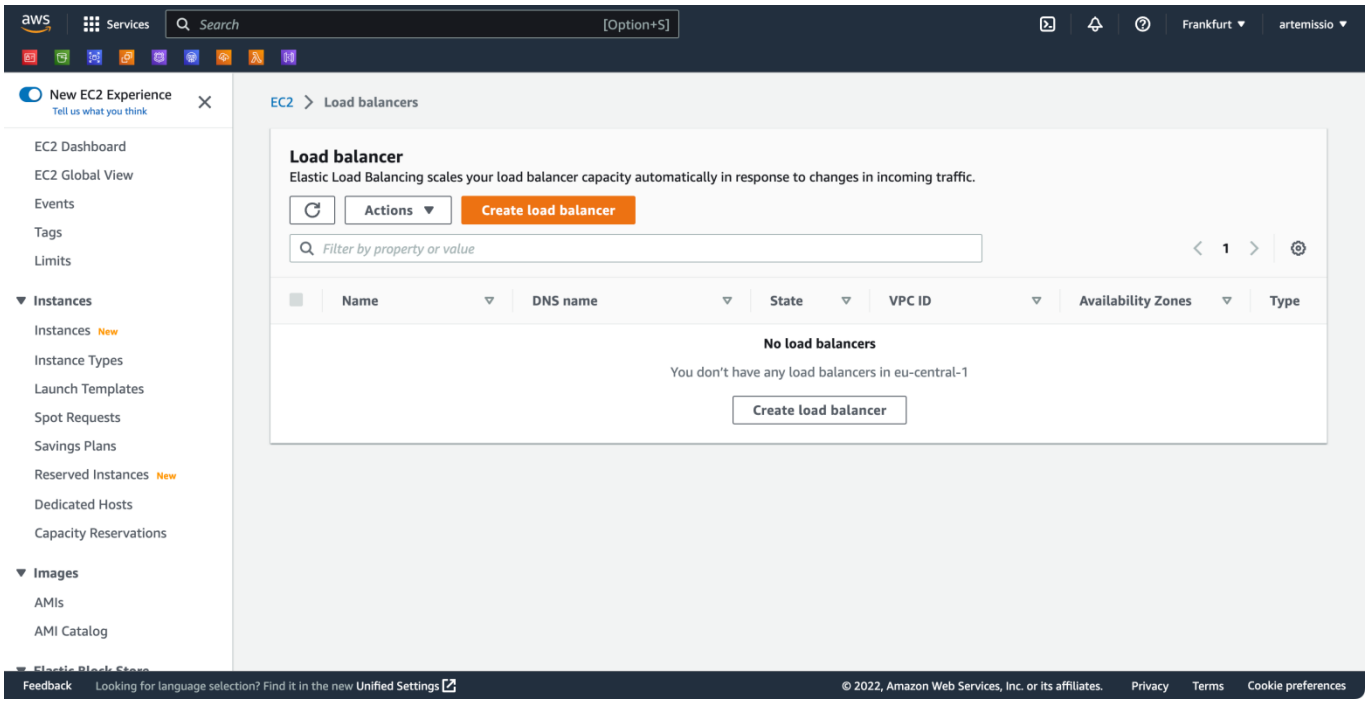

Рисунок 5.20 – UI консолі управління балансувальниками навантажень сервісу ELB

5.5.5 Сервіс AWS Elastic Beanstalk

AWS Elastic Beanstalk – сервіс, який дозволяє користувачам користувачам швидко розгортати програми та керувати ними у хмарі AWS, не вивчаючи інфраструктуру, яка їх запускає. Elastic Beanstalk полегшує управління, необмежуючи можливості вибору та контролю. Достатньо просто завантажити програму, і Elastic Beanstalk автоматично обробляє деталі надання сховища даних, балансування навантаження, масштабування та моніторинг працездатності програми. Elastic Beanstalk підтримує програми, розроблені на Go, Java, .NET, Node.js, PHP, Python і Ruby. Під час розгортання програми, Elastic Beanstalk створює вибрану підтримувану версію платформи та надає один або кілька ресурсів AWS, наприклад екземпляри Amazon EC2, для запуску програми.

Також можна виконувати більшість завдань із розгортання, наприклад змінювати розмір свого стеку екземплярів Amazon EC2 або контролювати свою програму, безпосередньо з веб-інтерфейсу (консолі) Elastic Beanstalk. Щоб використовувати Elastic Beanstalk, користувач створює програму, завантажує версію

програми у формі вихідного комплекту програми (наприклад, файл Java .war або C# .dll) до Elastic Beanstalk, а потім вказує детальну інформацію про програму. Elastic Beanstalk автоматично запускає середовище та створює та налаштовує ресурси AWS, необхідні для запуску коду. Після запуску середовища користувач зможе керувати ним і розгортати нові версії програми. Робочий процес Elastic Beanstalk зображено на рисунку 5.21.

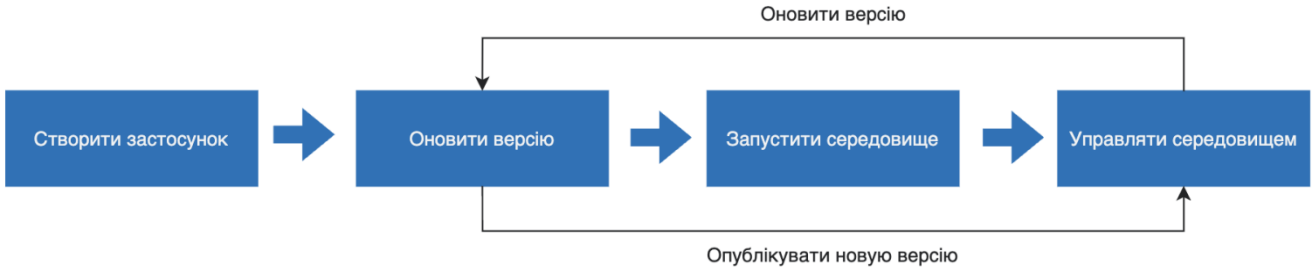

Рисунок 5.21– Робочий процес Elastic Beanstalk

Після створення та розгортання програми інформація про програму, включно з показниками, подіями та статусом середовища, стає доступною через консоль Elastic Beanstalk, API або інтерфейси командного рядка, включаючи уніфікований CLI AWS.

Даний сервіс буде використаний у роботі в якості зручного інструмента швидкого розгортання веб-застосунків. Саме завдяки даному сервісу у хмарному середовищі AWS будуть опубліковані головний веб-сайт, бек-офіс та консоль управління.

5.5.6 Сервіс AWS SES

Amazon SES (Simple Email Service) – це постачальник хмарних послуг електронної пошти, який можна інтегрувати в будь-яку програму для масових розсилок. Незалежно від того, чи надсилає користувач транзакційні чи маркетингові електронні листи, користувачі платять лише за те, що вони використовують. Amazon

SES також підтримує різні варіанти розгортання, включаючи виділені, загальні або приватні IP-адреси.

Створення великомасштабного рішення для роботи з електронною поштою часто є складним та дорогим завданням для бізнесу. Розробники повинні вирішувати питання інфраструктури, такі як керування сервером електронної пошти, конфігурація мережі та репутація IP-адреси. Крім того, багато сторонніх рішень для електронної пошти вимагають контрактів та переговорів про ціну, а також великі початкові витрати. Завдяки сервісу AWS SES досить просто додати бібліотеку до своєї програми, і її можна використовувати для будь-яких цілей.

Даний сервіс використовується інфраструктурною частиною фреймворку для надсилання користувачам листів із підтвердженням реєстрації, створення ERP систем та інших корисних оповіщень.

На рисунку 5.22 зображений UI панелі сервісу SES.

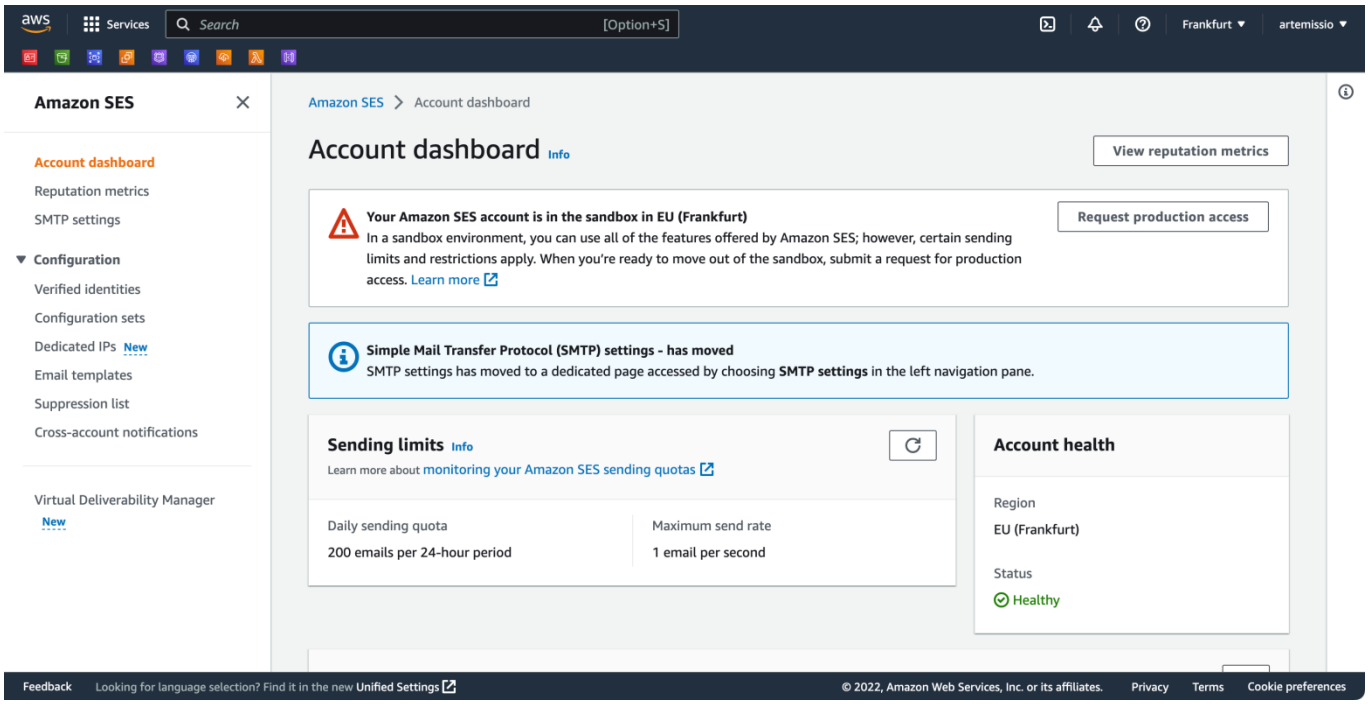

Рисунок 5.21 – UI консолі управління сервісом SES

5.5.7 Сервіс AWS Lambda

AWS Lambda – це безсерверний сервіс обчислень, що керується подіями, який дозволяє запускати код практично в будь-якому типі програм або серверних служб без виділення серверів або керування ними. Запускати Lambda можна з більш ніж 200 сервісів AWS та програм «Програмне забезпечення як послуга» (SaaS) і платити лише за те, що використовується.

Lambda запускає код у високодоступній обчислювальній інфраструктурі та виконує всі функції управління вашими обчислювальними ресурсами, включаючи обслуговування сервера та операційної системи, виділення ресурсів, автоматичне масштабування та ведення журналу. Використовувати Lambda можна для запуску коду будь-якого типу програм або серверних служб.

Розробники організують свій код за допомогою лямбда-функцій. Lambda запускає функцію лише тоді, коли це потрібно, і автоматично масштабується від кількох запитів на день до тисяч запитів на секунду. Користувачі платять тільки за обчислювальний час, що використовується, а не за той час, коли код не працює.

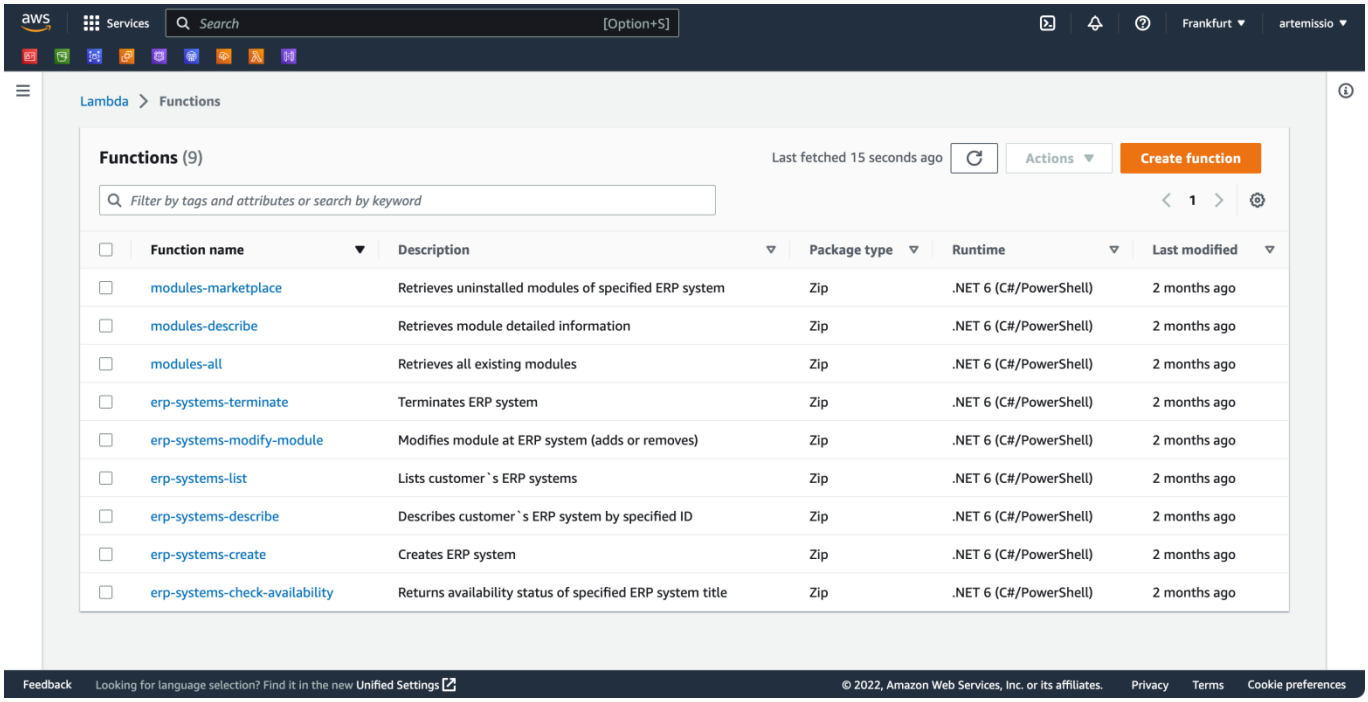

Рисунок 5.22 – Перелік створених Lambda функцій у консолі AWS

5.5.8 Сервіс AWS API Gateway

Amazon API Gateway – це повністю керований сервіс, який дозволяє розробникам легко створювати, публікувати, підтримувати, відстежувати та захищати API будь-якого масштабу. API-інтерфейси діють як «вхідні двері» для доступу додатків до даних, бізнес-логіки або функцій із серверних служб. Шлюз API дозволяє створювати REST та WebSocket API-інтерфейси, які забезпечують двосторонній зв'язок у режимі реального часу. Шлюз API підтримує контейнерні безсерверні робочі навантаження та веб-додатки. Шлюз API виконує всі завдання, пов'язані з прийомом та обробкою до сотень тисяч одночасних викликів API, включаючи керування трафіком, підтримку CORS, автентифікацію та контроль доступу, регулювання, моніторинг та контроль версій API.

У даному фреймворку шлюз API працюватиме в парі із сервісом AWS Lambda: API шлюз буде приєднаний до субдомену даного фреймворку і, використовуючи маппінг, буде перенаправляти запити до потрібної лямбда-функції залежності від шляху запиту. Даний механізм зображений на додатку В. Самі шлюзи, налаштовані через консоль сервісу, зображені на рисунку 5.23.

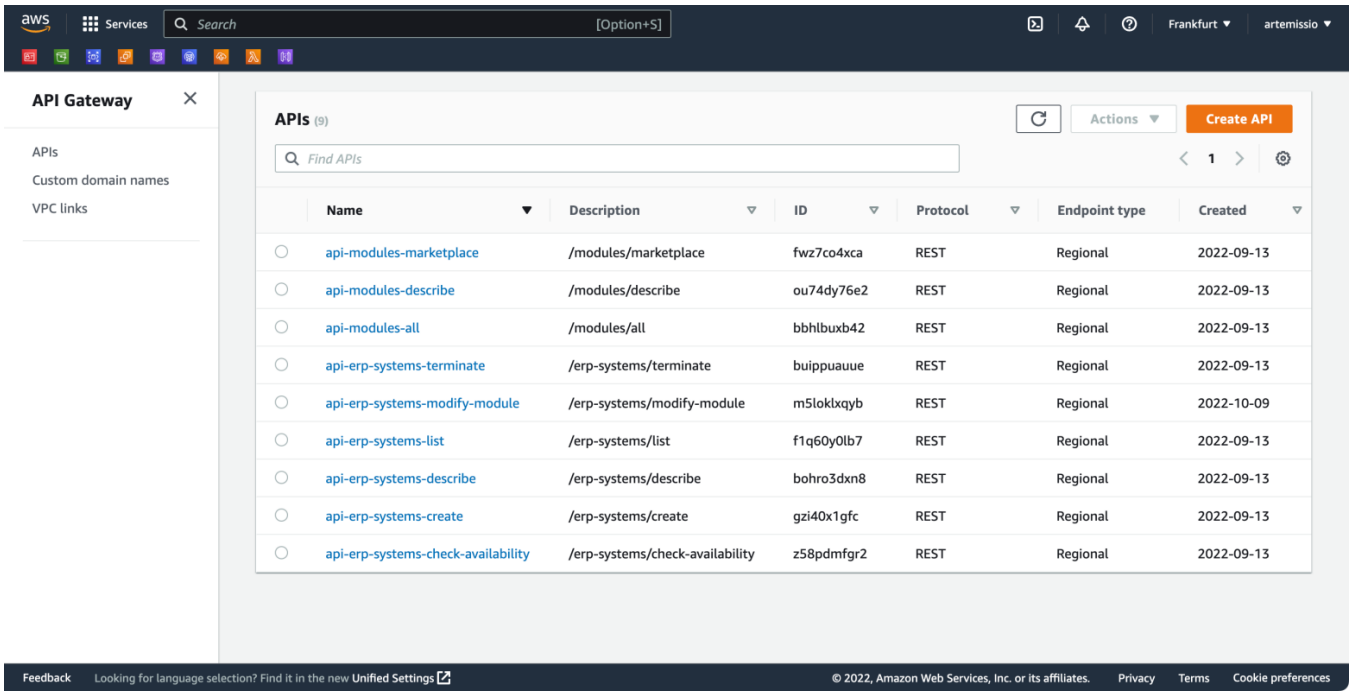

Рисунок 5.23 – Шлюзи, налаштовані через API Gateway

5.5.9 Сервіс AWS Route53

Amazon Route 53 – це високо-доступний та масштабований веб-сервіс системи доменних імен (DNS). Маршрут 53 пов'язує запити користувачів з інтернет-програмами, що працюють на AWS або локально. Використовувати Route53 можна для будь-якої комбінації трьох основних функцій: реєстрації домену, DNS-маршрутизації та перевірки працездатності.

Прикладами використання даного сервісу є наступні:

– керування мережним трафіком глобально. Служба використовує прості у використанні глобальні функції DNS, щоб забезпечити можливість створення, візуалізації та масштабування складних взаємозв'язків маршрутизації між записами та політиками;

– створення високо-доступних програм. Можна налаштувати політики маршрутизації для визначення та автоматизації реагування на несправності, наприклад перенаправлення трафіку в альтернативну зону доступності або регіон.;

– приватні параметри DNS. За допомогою цієї послуги можна призначати власні доменні імена та отримувати до них доступ у своїй віртуальній приватній хмарі Amazon (VPC). Також можна використовувати внутрішні ресурси та сервери AWS, не розкриваючи даних DNS у загальнодоступному Інтернеті.

Даний сервіс є дуже важливим і також є однією із причин вибору саме AWS в якості постачальника хмарних послуг. Цей сервіс бере участь у інфраструктурній частині фреймворку. Саме в ньому був створений глобальний інтернет домен даного фреймворку (erp-aurora.com), а також субдомени для API (api.erp-aurora.com), бек-офісу (backoffice.erp-aurora.com) та консолі управління (console.erp-aurora.com). Окрім цього, саме використовуючи цей сервіс, користувачі отримуватимуть аналогічні домени для створюваних ERP систем.

На рисунку 5.24 можна побачити UI консолі управління даного сервісу на прикладі подачі деталей про зареєстрований домен для фреймворку та іншої пов’язаної інформації.

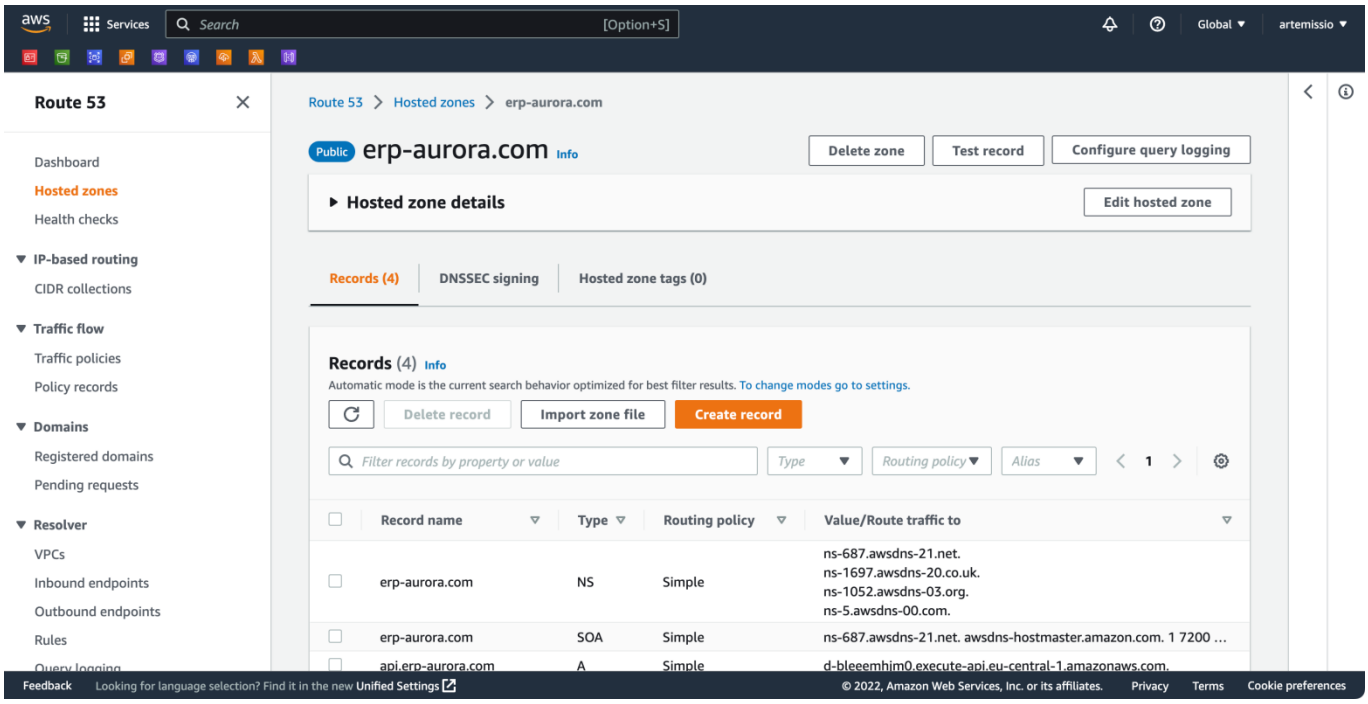

Рисунок 5.24 – UI консолі управління доменом сервісу AWS Route53

5.5.10 Сервіс AWS Event Bridge

AWS Event Bridge – це безсерверний сервіс, який використовує події для з'єднання компонентів програми, спрощуючи створення масштабованих програм, керованих подіями. Його можна використовувати для маршрутизації подій з таких джерел, як домашні програми, сервіси AWS та інше програмне забезпечення, до споживчих програм у організації. Event Bridge забезпечує простий та узгоджений спосіб отримання, фільтрації, перетворення та доставки подій.

Event Bridge отримує події, які є індикаторами змін у середовищі, та застосовує правила для маршрутизації подій до цілей. Правила зіставляють події з цілями з урахуванням структури подій, званої шаблоном чи розкладом подій. Наприклад, ви можете створити правило, яке надсилає подію у вашу функцію Lambda, коли інстанс Amazon EC2 переходить з режиму очікування в режим роботи. Усі події, що надходять на міст подій, підключаються до події. Правило прив'язане до однієї шині подій, тому його можна застосовувати лише до подій у цій шині подій. Кожний обліковий запис має стандартну шину, яка отримує події від сервісів AWS, і

користувачі можуть створювати власні шини подій для надсилання та отримання подій з різних облікових записів або регіонів.

Даний сервіс використовується на даний момент виняткова для формування фінансових звітностей. Завдяки ньому було створене налаштування для запуску лямбда-функції за розкладом (1 числа кожного місяця), яка буде опрацьовувати дані у БД фреймворку, формувати файли звітності та завантажувати їх до S3 бакету. Пізніше ці файли можна використати для проведення статистичного аналізу або інших опрацювань.

На рисунку 5.25 зображена панель управління шинами подій сервісу AWS Event Bridge.

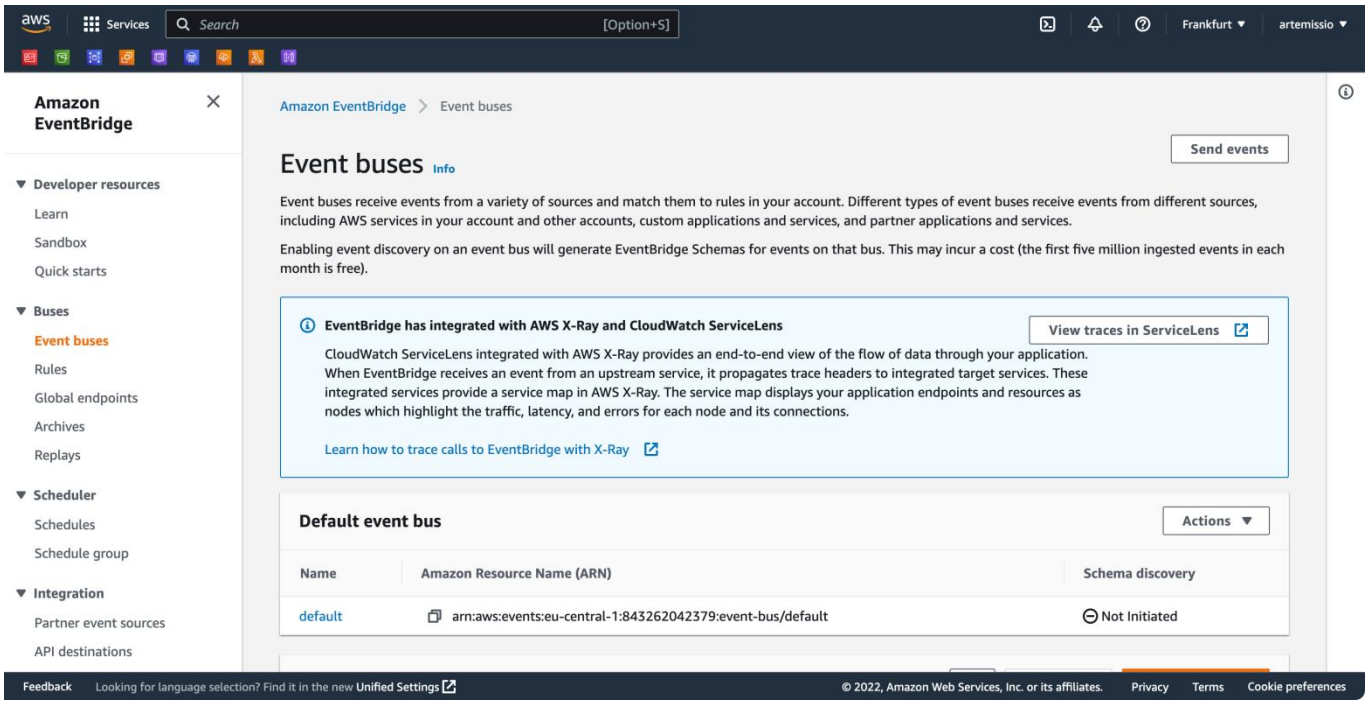

Рисунок 5.25 – Панель управління шинами подій сервісу AWS Event Bridge

5.5.11 Загальна схема взаємодії хмарних сервісів

Схема взаємодії хмарних сервісів зображена на додатку Г. Як можна побачити, своєрідною вхідною точкою є сервіс Route53, оскільки саме в ньому створений глобальний інтернет домен, на який будуть надходити запити. Далі, в залежності від

шляху, на який був відправлений запит, він перенаправляється на один із декількох інфраструктурних додатків або до ERP системи:

– erp-aurora.com – запит до головного веб-сайту. Цей додаток опублікований за допомогою сервісу Elastic Beanstalk. Він взаємодіє із S3 (для зчитування шаблонів електронних листів), RDS (сервісом БД) та SES (відправка електронних листів);

– backoffice.erp-aurora.com – запит до бек-офісу фреймворку. Цей додаток, аналогічно до головного веб-сайту, опублікований за допомогою сервісу Elastic Beanstalk. Він також взаємодіє із S3 (для завантажування файлів модулів та зчитування статистики), RDS (сервісом БД) та SES (для моніторингу статистики користування даним сервісом);

– api.erp-aurora.com – запит до API фреймворку. API опублікована за допомогою сервісу API Gateway, який перенаправляє запити до потрібної Лямбда-функції, які в свою чергу також взаємодіють із RDS, S3, SES;

– console.erp-aurora.com – запит до консолі управління. Цей додаток також опублікований у сервісі Elastic Beanstalk, проте не взаємодіє із БД фреймворку та іншими інфраструктурними сервісами. Він створений в якості веб-інтерфейсу для взаємодії із створеними ERP системами.

Event Bridge регулюється самим AWS, незалежно від інших сервісів, та взаємодіє із однією єдиною лямбда-функцією, яка буде формувати фінансову звітність та статистику за минулий місяць, і зберігатиме її у S3.

Самі ERP системи з точки зору хмарних сервісів являють собою логічне об’єднання трьох сервісів – EC2 (віртуальний сервер (або декілька), на якому працює веб-сервіс), Elastic Load Balancing (для розподілення навантаження між серверами) та RDS (БД ERP системи).

5.6 Програмні моделі інфраструктурних додатків

Як і будь-які веб-застосунки, фреймворку також потребується БД для збереження інформації про його користувачів та інші ресурси. Оскільки дані в фреймворку будуть строго структуровані, використовується реляційна БД (також із

використанням сервісу RDS). ER діаграма інфраструктури фреймворку зображена на додатку Д.

Як можна побачити ключовими таблицями є таблиця користувачів фреймворку (Accounts), ERP систем (ERP systems) та модулів (Modules). Використовуючи дану схему відношень, можна легко знайти ERP системи користувача, модулі, встановлені на ERP системі або які хмарні ресурси (екземпляри віртуальних серверів, баз даних або домени) використовуються тією чи іншою ERP системою.

5.7 Розробка головного веб-сайту

Згідно технічних вимог головний веб-сайт повинен слугувати відразу як ознайомчий сайт, на якому користувач може ознайомитися з усіма послугами даного фреймворку (модулями (їхніми описами та детальною інформацією), веб-сервісом ERP системи), так і функціональний веб-застосунок.

При переході на потрібну сторінку, користувач може знайти необхідну інформацію про модулі (рисунок 5.26), власну сторінку (рисунок 5.27).

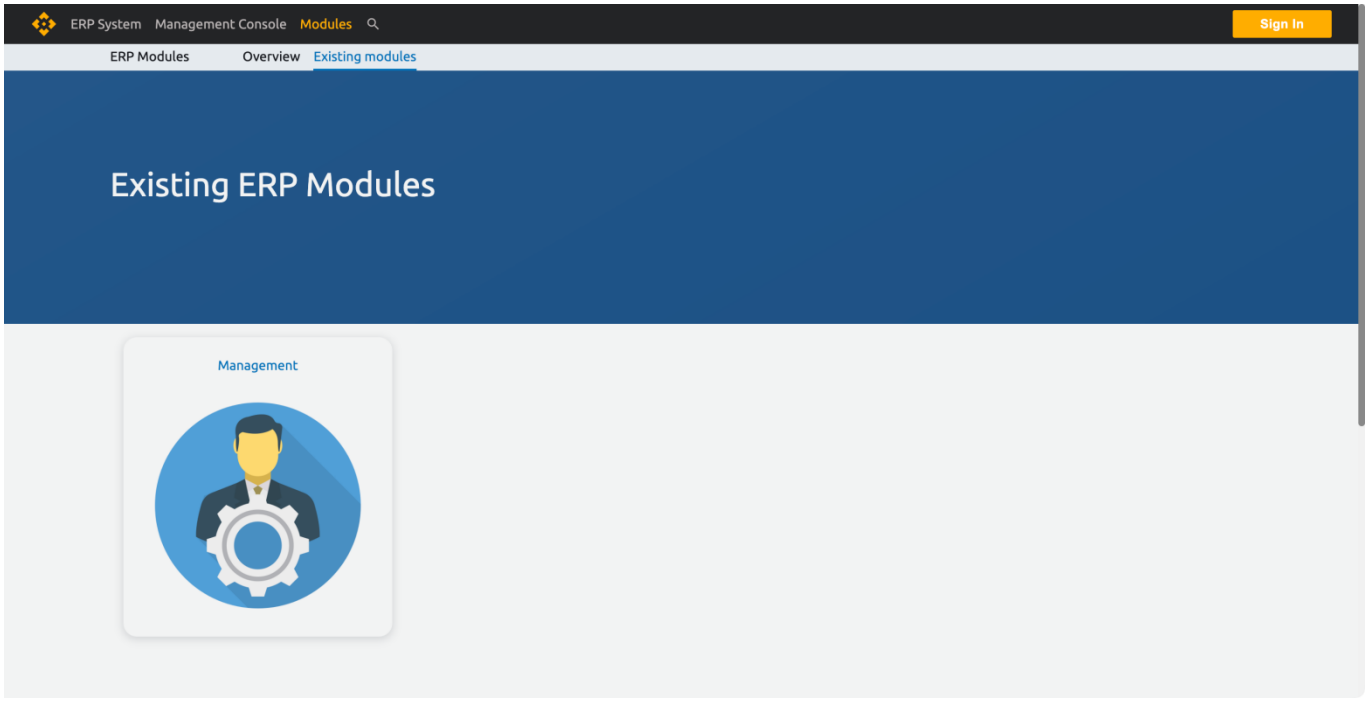

Рисунок 5.26 – Сторінка із існуючими у фреймворку модулями

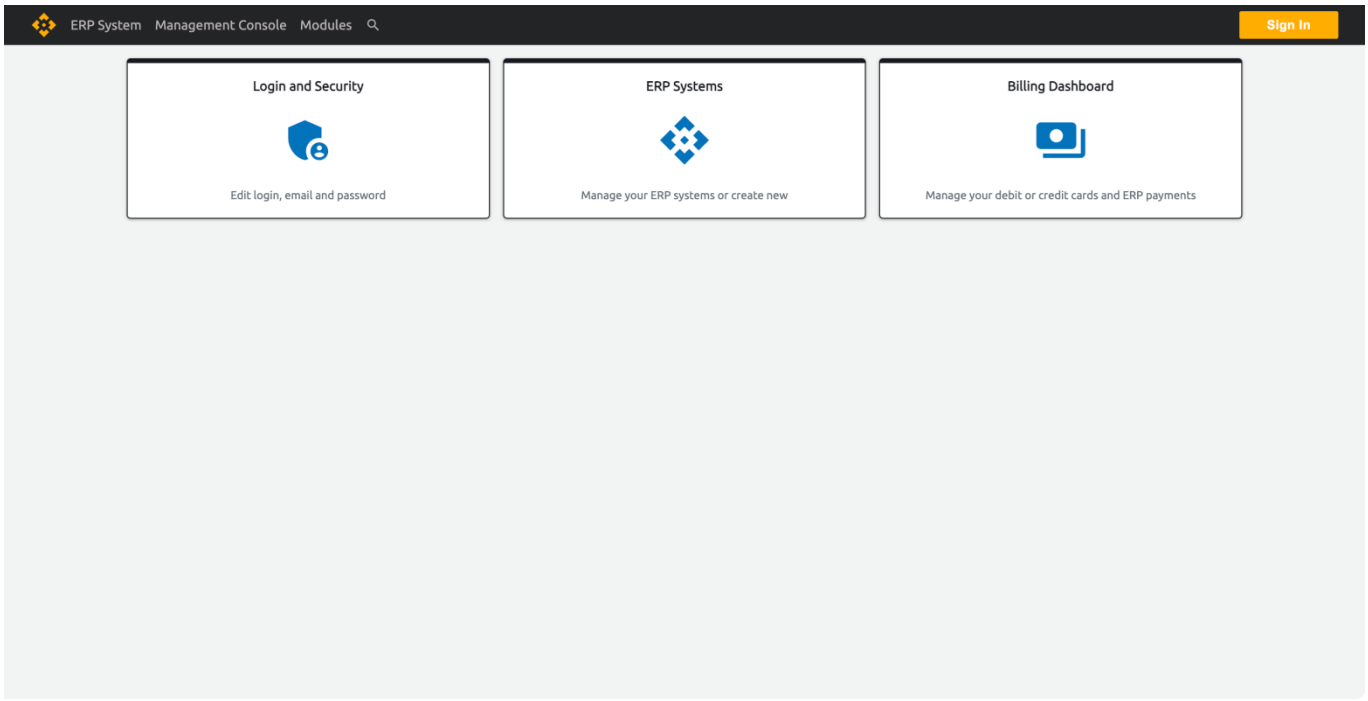

Рисунок 5.27 – Сторінка облікового запису користувача

На вкладці ERP Systems користувач знайде всі свої ERP системи та може створити нові, використовуючи надзвичайно прості налаштування – введення назви системи, вибрати необхідні модулі та вказати налаштування БД (рисунок 5.28 – 5.29).

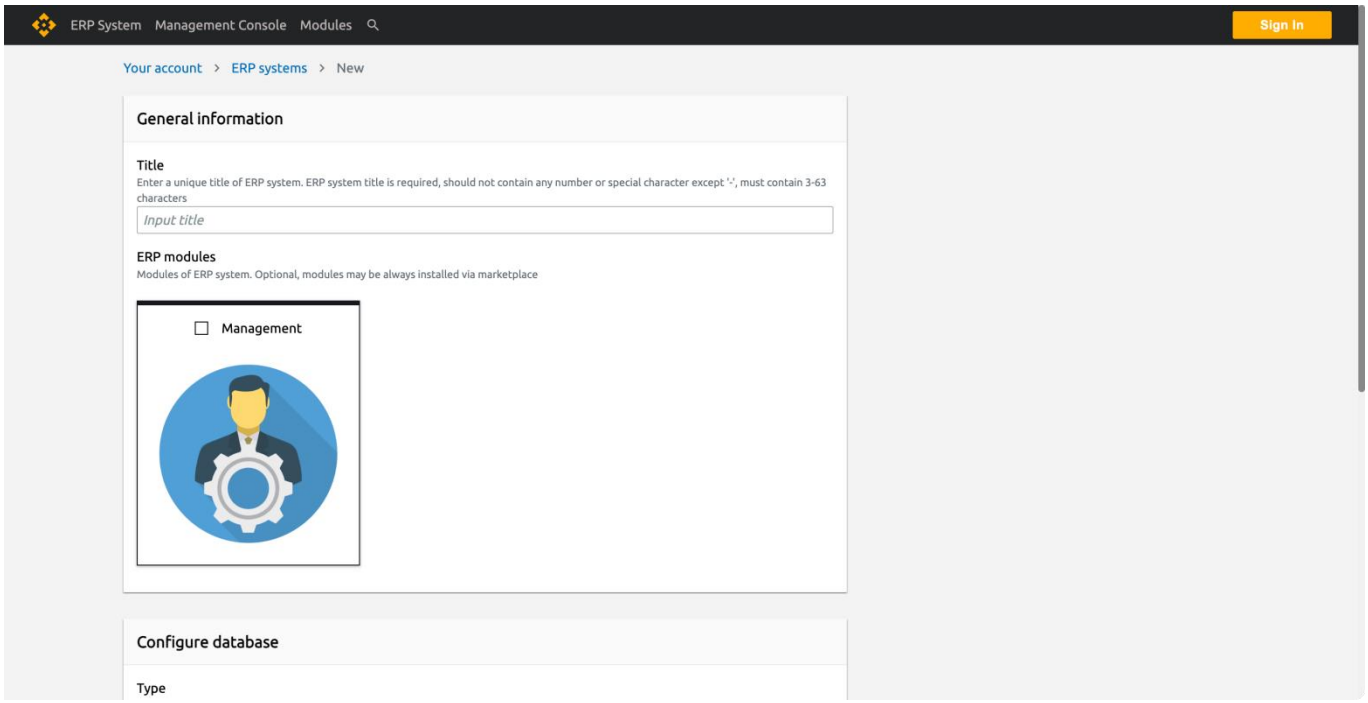

Рисунок 5.28 – Створення ERP системи (введення назви та вибір модулів)

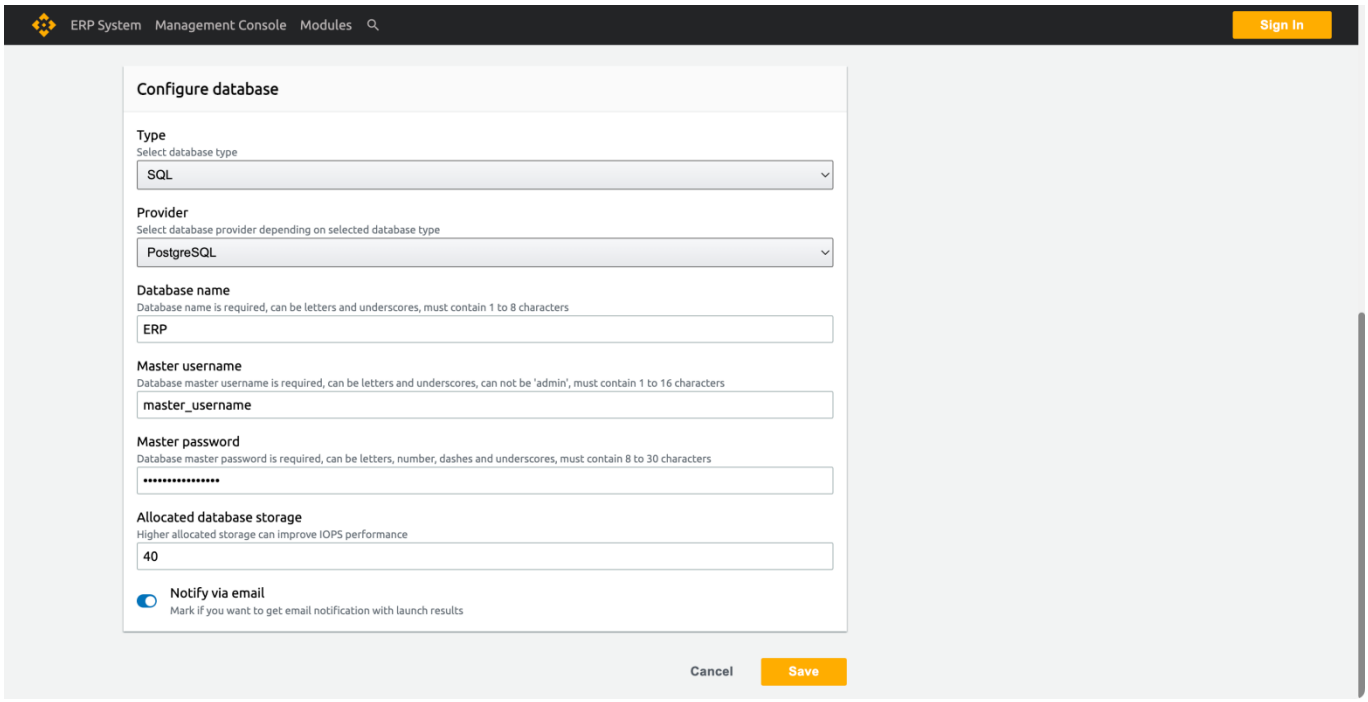

Рисунок 5.29 – Створення ERP системи (налаштування БД)

В ході створення можна відмітити, чи хоче користувач отримати електронного листа про результат запуску (як успішний, так і у випадку помилки). В результаті успішної операції на електронну адресу користувача прийде лист із результатами (рисунок 5.30).

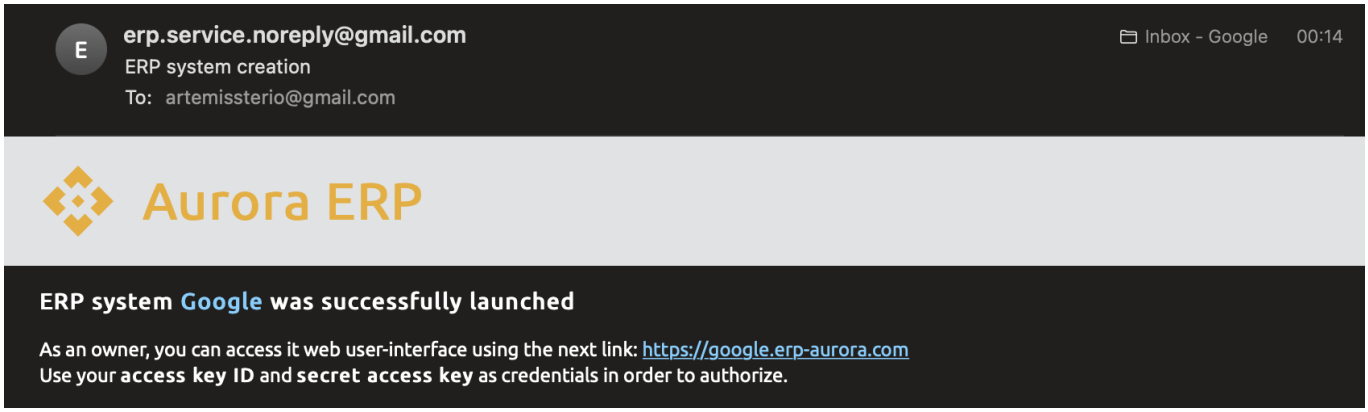

Рисунок 5.30 – Електронний лист про успішний запуск ERP системи

Процес запуску ERP системи зображений у вигляді діаграми на додатку Ж. Як можна побачити, він складається із декількох наступних кроків:

– запис інформації про нову систему до БД. Після цього на вкладці ERP систем можна побачити, на якому етапі створення знаходиться вибрана система (рисунок 5.31). Всього існує 4 стани: Initializing (запис нової системи до БД), Creating database (створення екземпляру БД системи), Creating instance (створення екземпляру веб-серверу та балансувальника навантаження), Creating domain (створення глобального домену системи), Running (вказує, що система знаходиться у робочому стані (рисунок 5.35)), Shutting down (даний статус система набуватиме під час її видалення), Terminated (вказує, що система була видалена та/або знаходиться у неробочому стані);

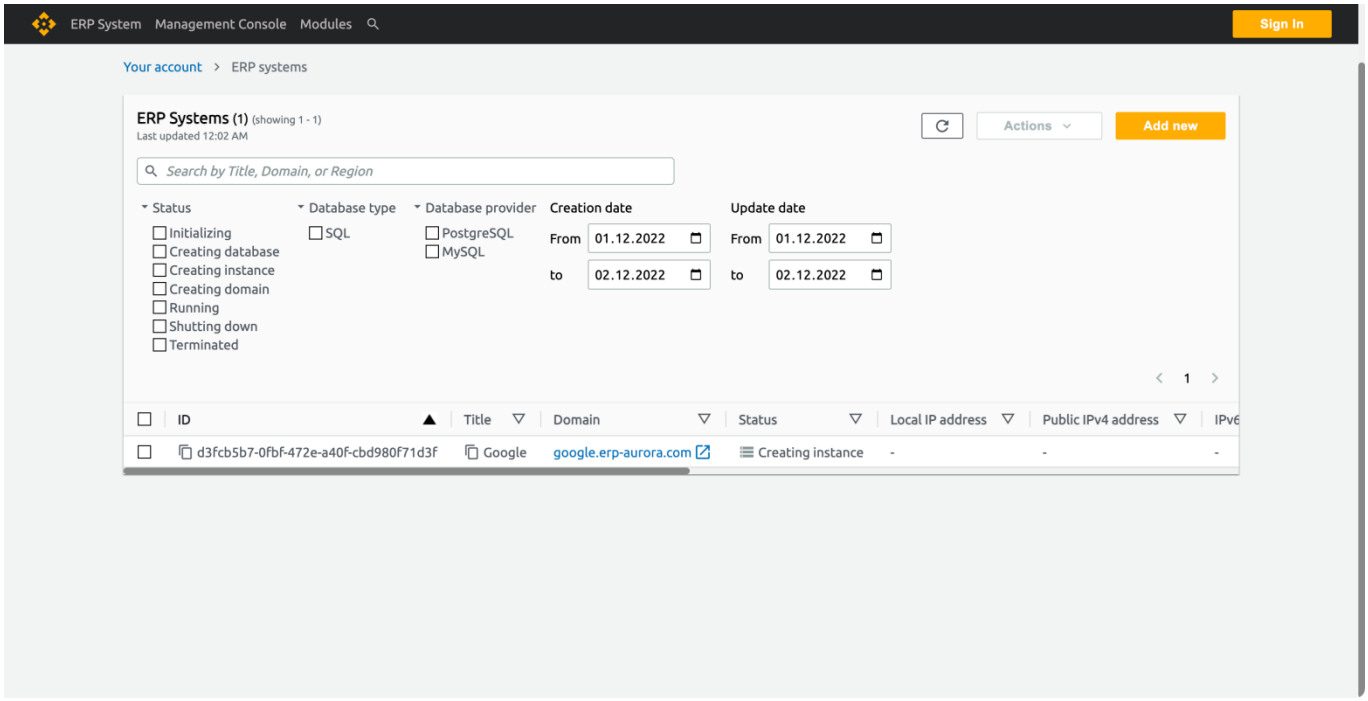

Рисунок 5.31 – Етапи створення ERP системи

– запуск БД системи в залежності від налаштувань. В першу чергу створюється БД система, адже саме до неї будуть перші запити від веб-серверу на створення таблиць в залежності від вибраних модулів. На рисунку 5.32 зображена панель налаштувань створеної БД у консолі AWS. Тільки після успішного запуску БД процес створення система перейде до наступного кроку;

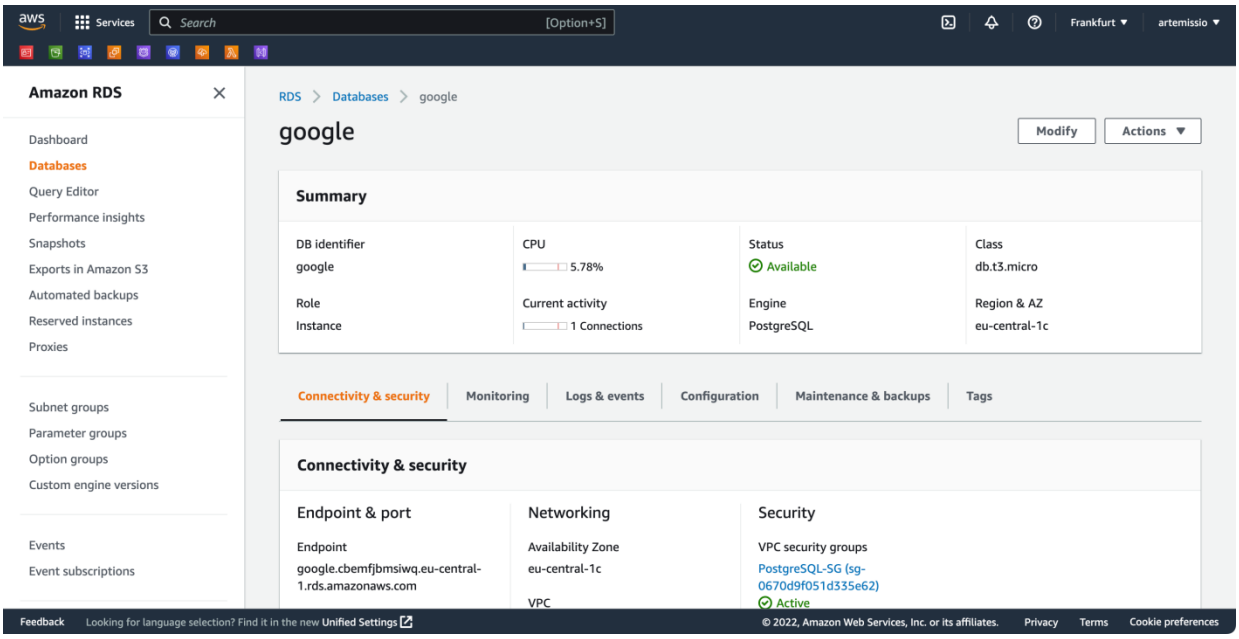

Рисунок 5.32 – Створена бази даних ERP системи

– запуск екземпляру веб-серверу. Після успішного запуску БД відбувається запуск віртуального серверу із скриптом на скачування виконуваного файлу для власне веб-сервісу і запускає його на 80 та 443 портах. Згаданий файл зберігається у S3 бакеті у приватному доступі, але є доступним для екземплярів веб-серверів EC2. На рисунку 5.33 можна побачити деталі створеного серверу у консолі AWS;

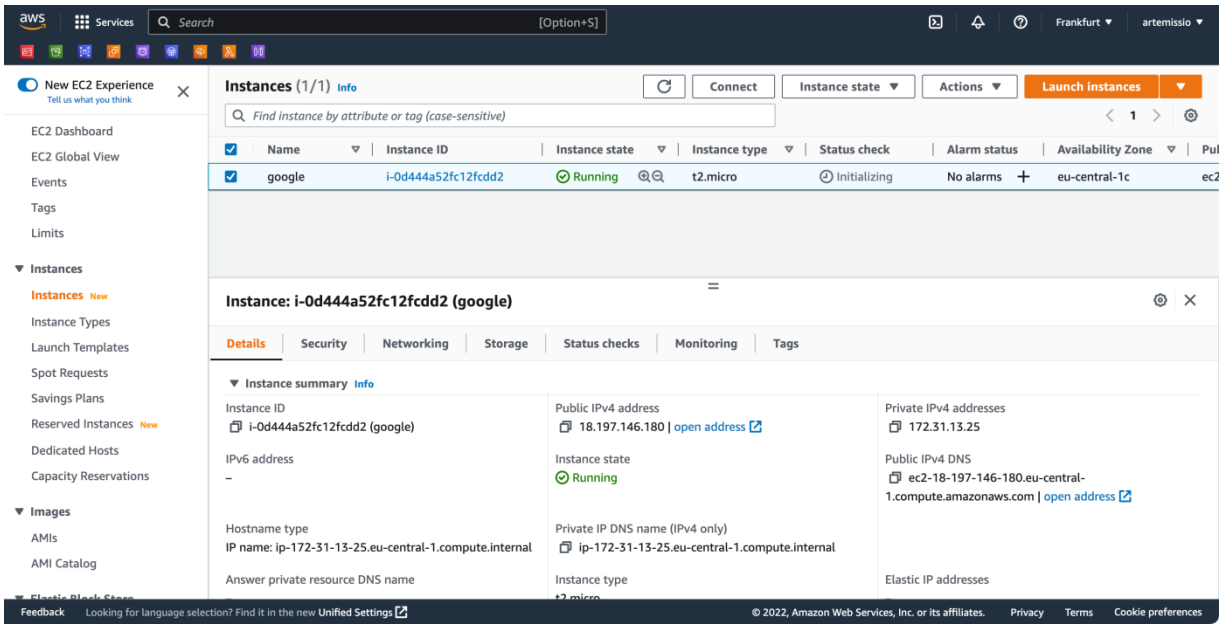

Рисунок 5.33 – Запущений веб-сервер ERP системи

– запуск балансувальника навантажень. Після успішного запуску веб-сервера відбувається запуск балансувальника навантажень із приєднаним веб-сервером (рисунок 5.34). Після ряду перевірок на роботоздатність серверу балансувальником процес переходить до наступного кроку;

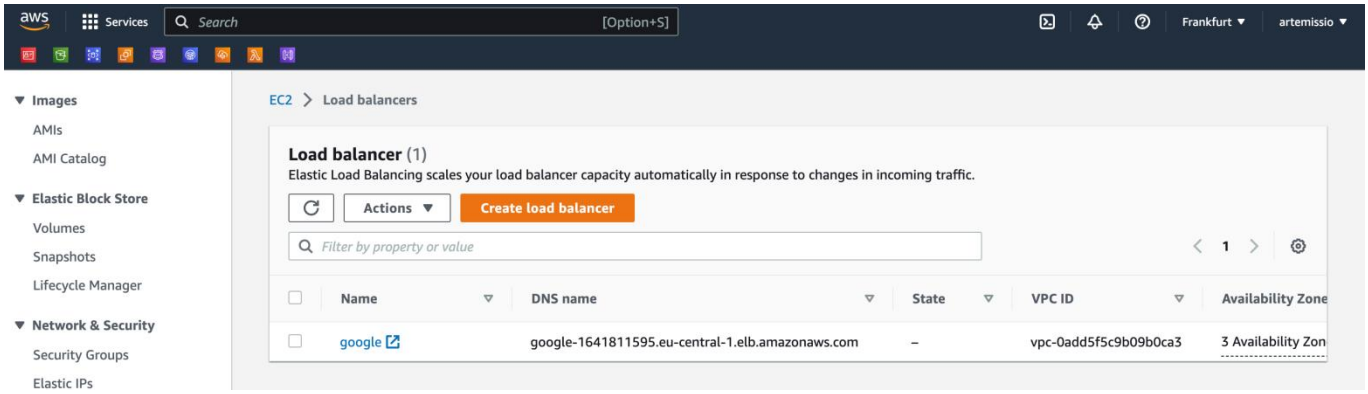

Рисунок 5.34 – Налаштований балансувальник навантаження ERP системи

– створення сабдомену і налаштування перенаправлення трафіку на створений балансувальник навантажень. На цьому процес запуску ERP системи завершується, система приймає статус Running (рисунок 5.35), її сабдомен перенаправляє трафік на веб-сервіс (рисунок 5.36).

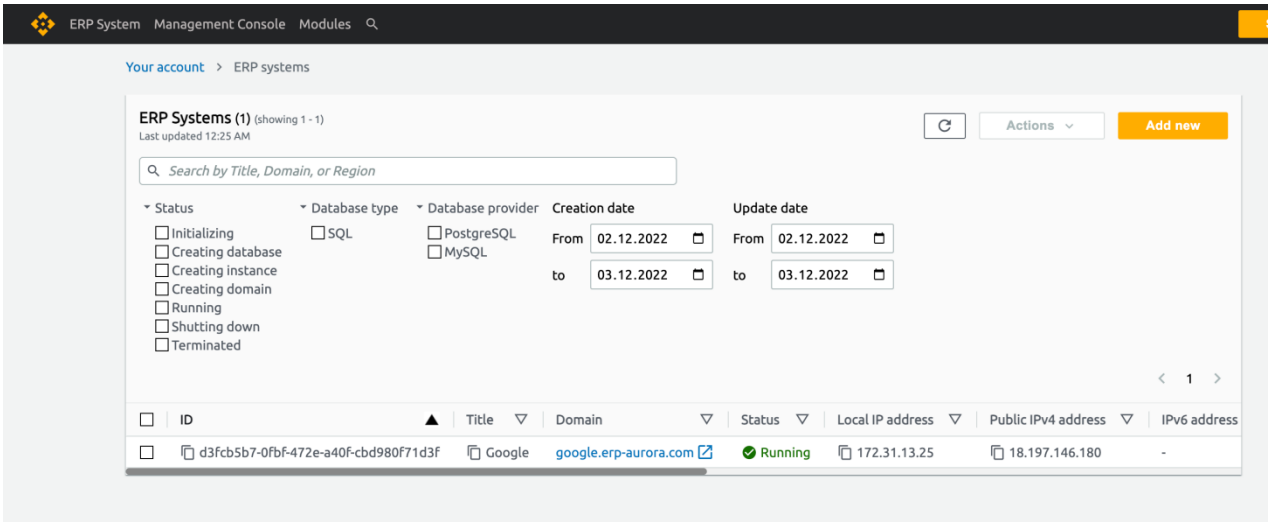

Рисунок 5.35 – Статус Running у запущеної ERP системи

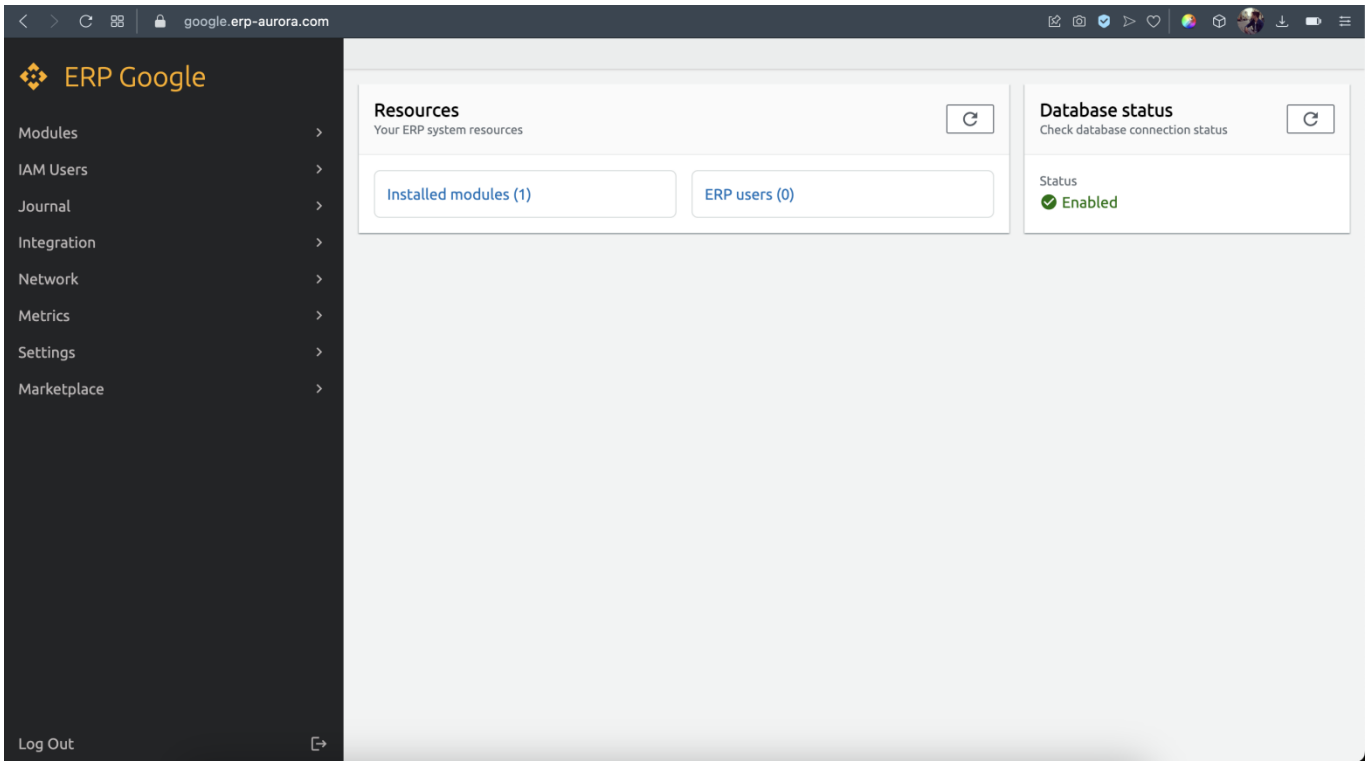

Рисунок 5.36 – Повністю запущена ERP система із глобальним інтернет доменом

Процес видалення ERP системи зображений на додатку И і є повністю зеркальним до процесу створення: система набуває статусу Shutting down, видаляється інтернет сабдомен, щоб не можна було надсилати до системи запити під час її видалення, потім балансувальник навантажень, після цього веб-сервер, і в нарешті – БД системи. В кінці процесу користувач отримує електронного листа, який засвідчує, що вказана ERP система була успішно видалена (рисунок 5.37), а система набуває статусу Terminated (рисунок 5.38).

Рисунок 5.37 – Електронний лист про успішне видалення ERP системи

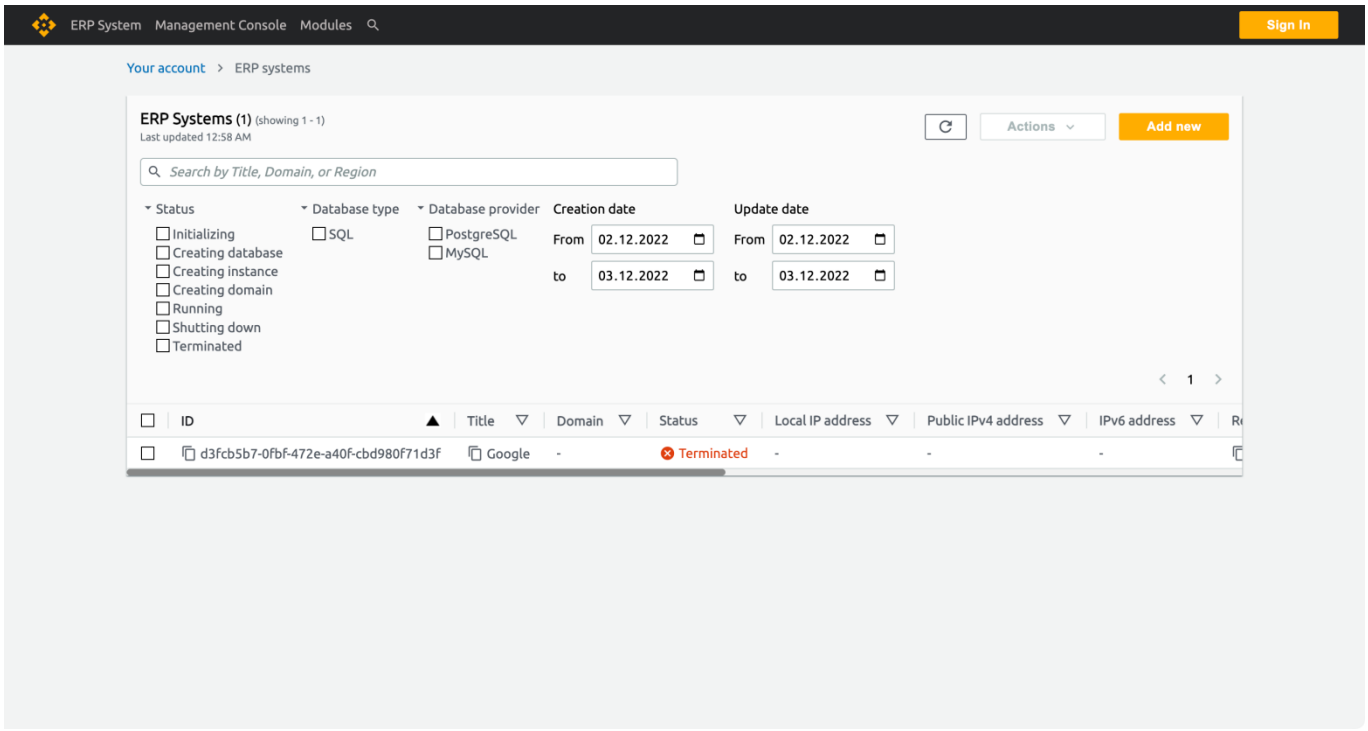

Рисунок 5.38 – Набуття ERP системою статусу Terminated

5.8 Розробка бек-офісу

Як було зазначено у вимогах до фреймворку, необхідно розробити окремий застосунок для адміністраторів фреймворку. Веб-додаток бек-офісу є саме даним інструментом. Як і всі інші застосунки, це Blazor server застосунок, розгорнутий у хмарі за допомогою сервісу Elastic Beanstalk у хмарі AWS.

Завдяки даному механізму адміністратори фреймворку матимуть можливість переглядати зареєстрованих у фреймворку користувачів, створені ERP системи, стан всіх інших інфраструктурних додатків (рисунок 5.39) а також бути в курсі про використання хмарних сервісів та проводити їхній моніторинг.

Однією з головних функцій, які надає даний застосунок – публікація модулів. Саме через нього усі нові модулі, що розроблятимуться, після ряду тестів та перевірок, будуть публікуватися для публічного ознайомлення користувачами та можливістю інсталяції до ERP систем. На рисунку 5.40 зображена форма для публікації модуля. Як можна побачити, при публікації необхідно вказати файл із модулем, його детальний опис та головний банер.

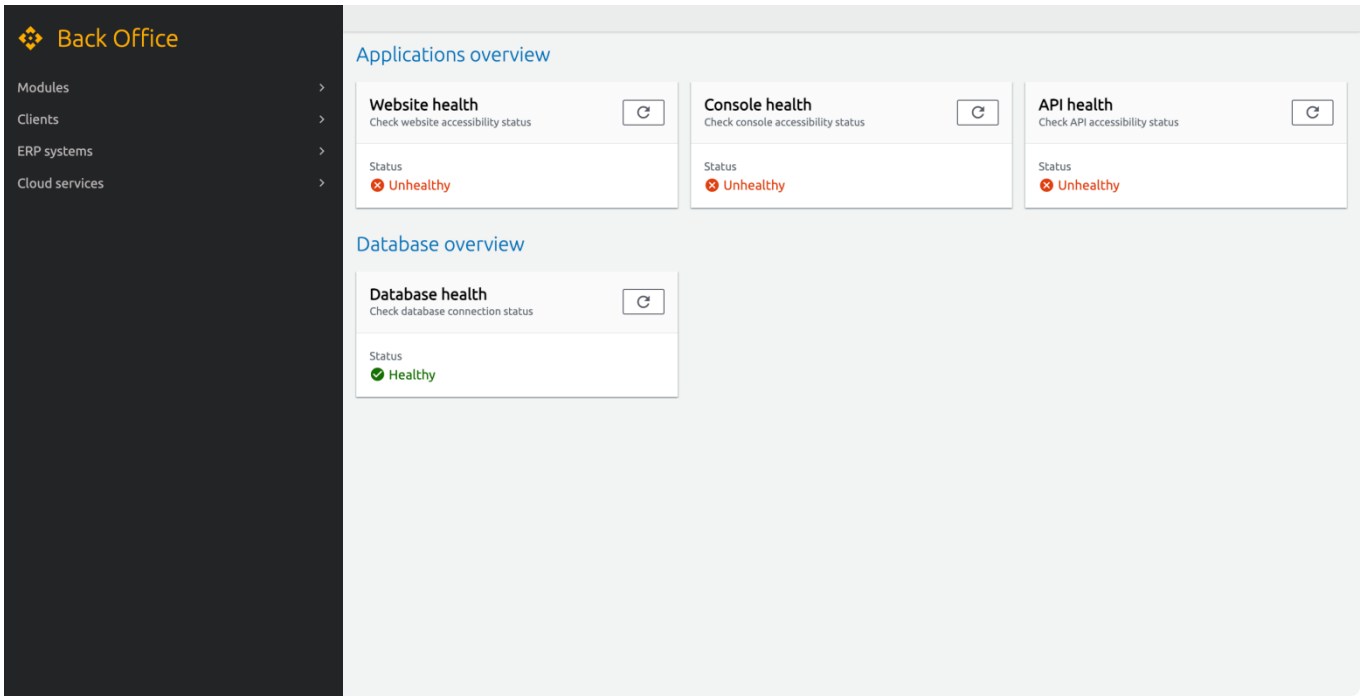

Рисунок 5.39 – Сторінка зі статусами інших застосунків

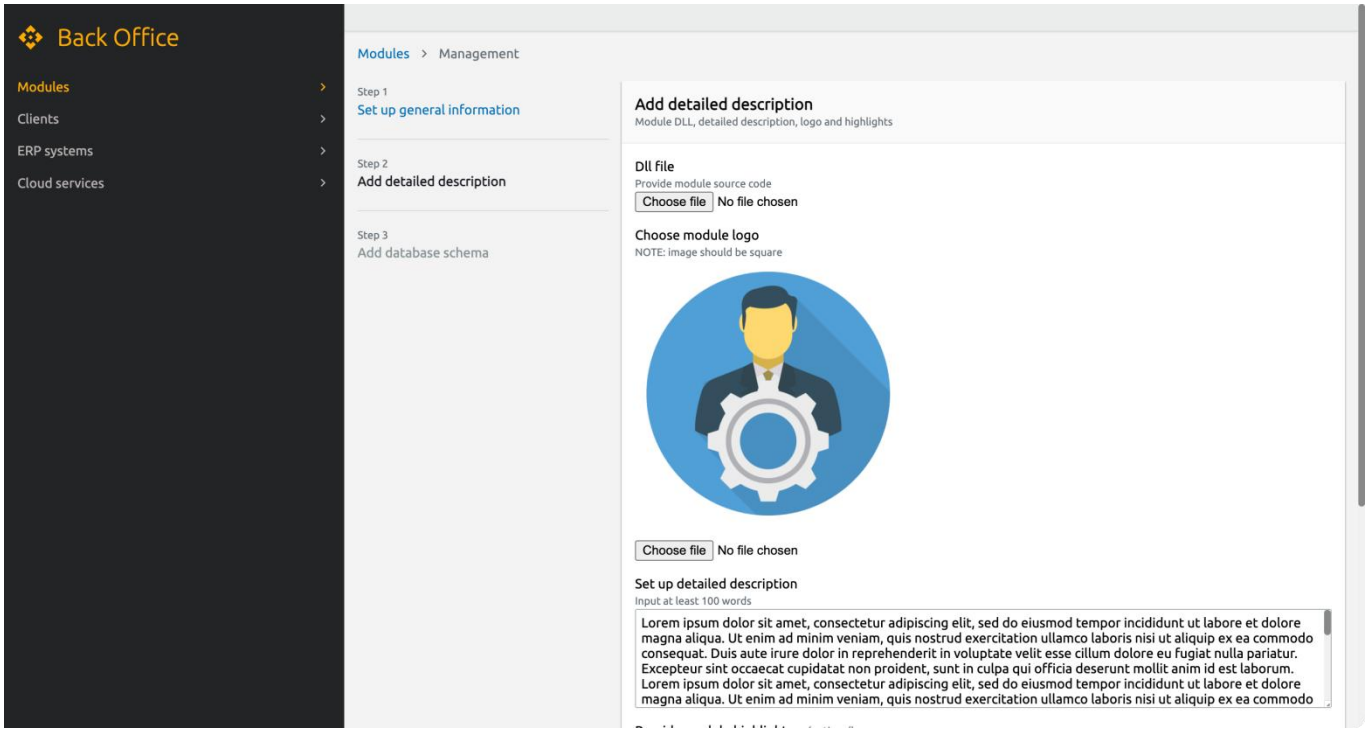

Рисунок 5.40 – Процес заповнення даними форми для публікації модуля

Після публікації модуль з’являтиметься на головному веб-сайті, де користувачі зможуть ознайомитися з його детальною інформацією.

5.9 Розробка API

Однією з вимог до розробки було створення стабільного API фреймворку, здатного обробляти запити у будь-який момент і який буде справлятися із будь-яким навантаженням. Із такою поставленою задачею чудово справиться технологія безсерверної архітектури.

API фреймворку представляє собою набір лямбда-функцій (рисунок 5.22), кожна із яких оброблятиме свій запит. Всього створено 9 лямбда-функцій:

– modules-marketplace: повертає всі невстановлені модулі у вказаній ERP системі;

– modules-describe: повертає детальну інформацію про вказаний модуль;

– modules-all: повертає всі опубліковані модулі фреймворку;

– erp-systems-list: повертає перелік ERP систем користувача;

– erp-systems-describe: повертає детальну інформацію про вказану ERP систему користувача;

– erp-systems-check-availability: повертає статус доступ назви ERP системи. Використовується для перевірки, щоб не можна було створити декілька систем з однаковими назвами;

– erp-systems-create: запускає процес створення ERP системи;

– erp-systems-modify-module: встановлює або видаляє модуль із ERP системи;

– erp-systems-terminate: запускає процес видалення ERP системи.

Сама API є API шлюзом сервісу API Gateway, до якої приєднані дані лямбда-функції, і яка приєднана до сабдомену API із відповідним мапінгом – своєрідна маршрутизація, яка вказує, на яку API надіслати запит в залежності від шляху (рисунок 5.41).

Дана реалізація є надзвичайно вигідною, оскільки оплата буде тільки за відпрацьовані запити, а також сама архітектура є дуже надійною – кожний запит буде оброблятися окремою функцією, тому навантаження як такого немає.

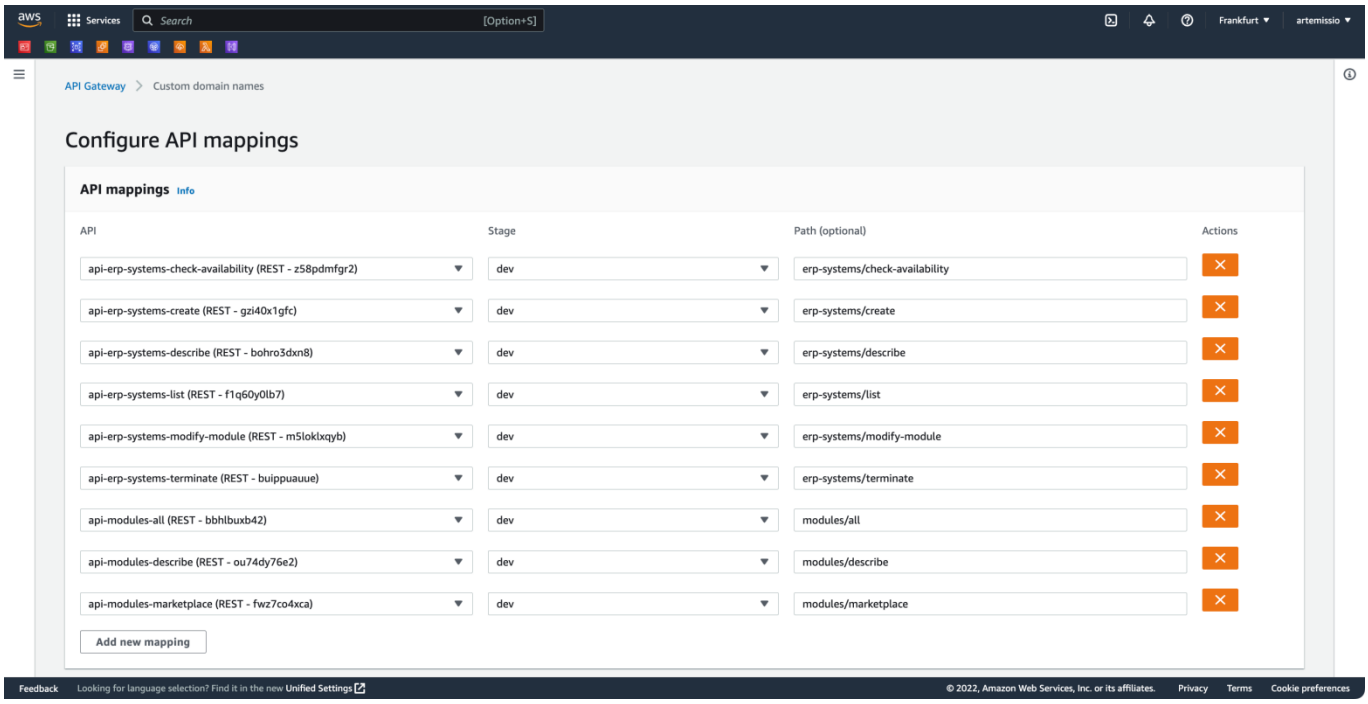

Рисунок 5.41 – Мапінг API шлюзу

5.10 Розробка веб-сервісу ERP системи

Веб-сервіс ERP системи, аналогічно до бек-офісу або головного веб-сайту, є Blazor застосунком. Цей додаток розроблятиметься та оновлюватиметься із часом, і кожна нова версія застосунку буде завантажуватися із S3 бакету під час запуску ERP системи.

При спробі переходу на будь-яку сторінку веб-сервісу відбувається перевірка авторизації користувача. Якщо він не авторизований, його буде перенаправлено на сторінку авторизації (рисунок 5.42), де він має ввести свої ключ доступу та секретний ключ. Доступ до веб-сервісу має лише власник ERP системи. Час дії сесії складає 12 годин: упродовж цього часу після закриття браузеру і відкритті сторінки знову користувачу не доведеться авторизуватися вдруге. Такий механізм забезпечить веб-сервіс від входу сторонніх користувачів та унеможливить крадіжку корпоративної інформації або інші шкідливі операції.

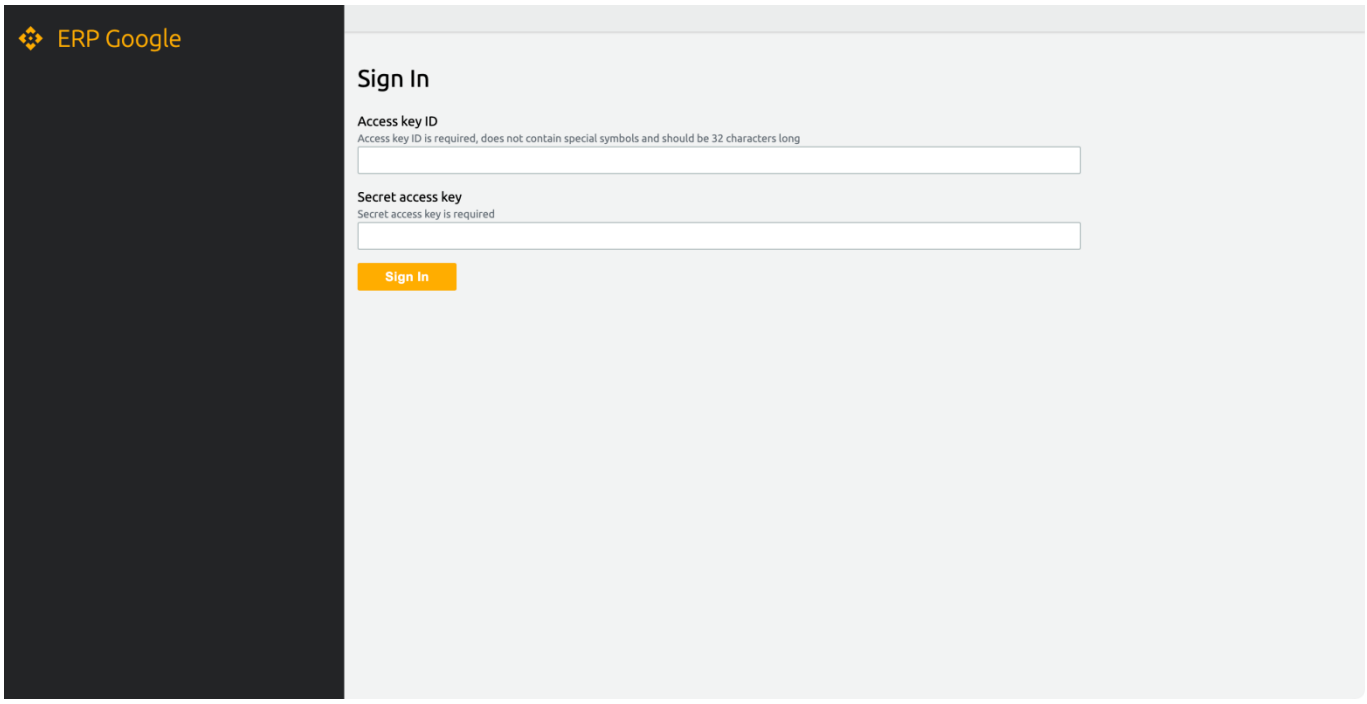

Рисунок 5.42 – Сторінка авторизації у веб-сервісі ERP системи

У випадку успішної авторизації користувача буде перенаправлено до головної сторінки, на якій відображається статус роботи БД, а також важливі ресурси системи: IAM користувачі та встановленні модулі.

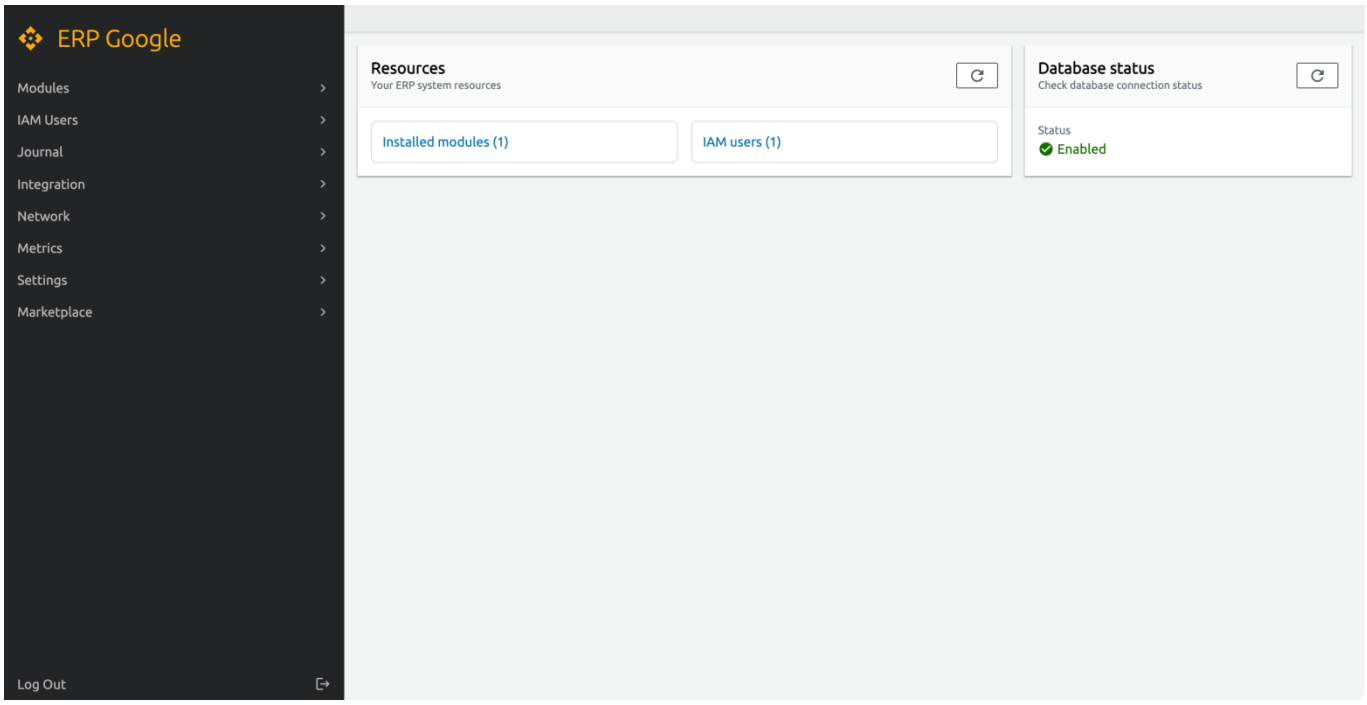

Рисунок 5.43 – Головна сторінка веб-сервісу

При переході на сторінку модулів користувачеві відображаються встановлені модулі у системі, при переході на кожний конкретний відображатимуться дані із таблиць саме цього модулю (рисунок 5.44). На рисунку 5.44, наприклад, це таблиця Відділів (Departments) модуля Управління (Management).

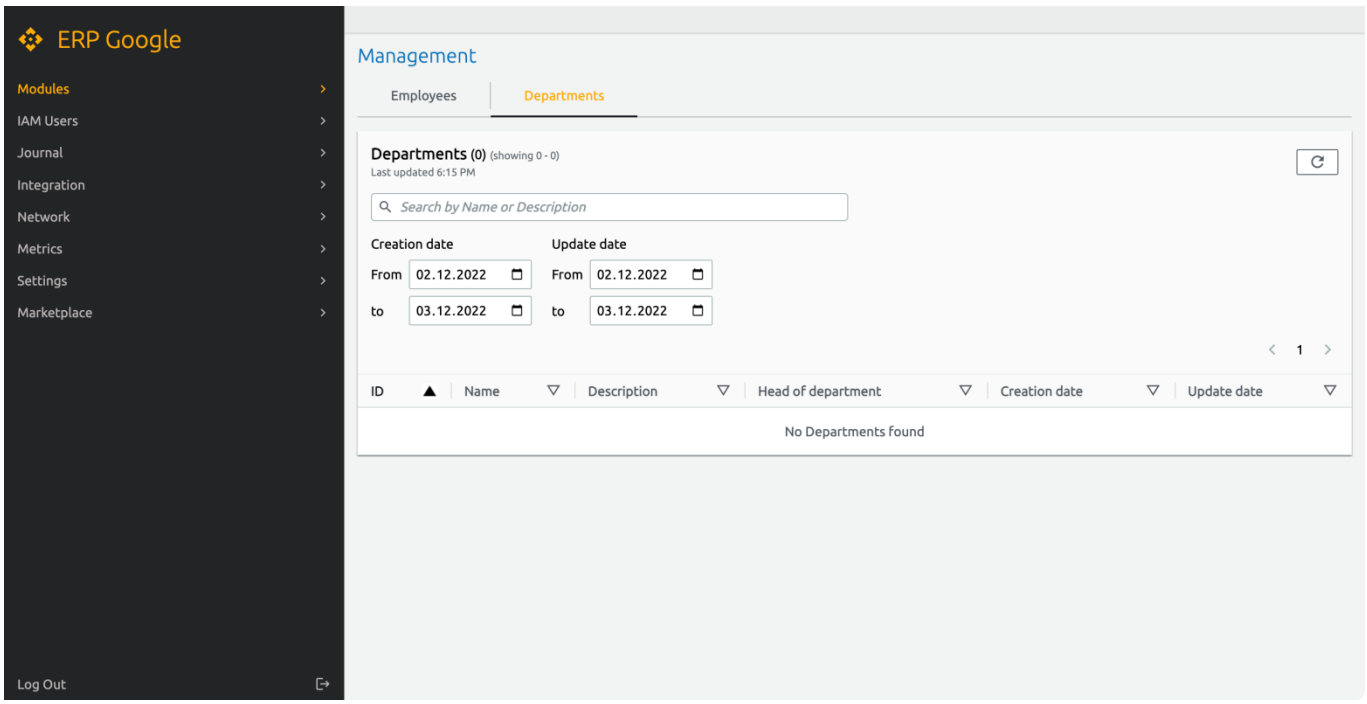

Рисунок 5.44 – Відображення даних таблиць конкретного модуля

Технологія IAM в даному сервісі реалізована в якості користувача із логіном, згенерованими автоматично ключем доступу та секретним ключем (рисунок 5.45), а також набором прав доступу (рисунок 5.46), які поділяються на 2 частини:

– загальні права доступу (Access settings): задає права доступу до БД (Database operations), операції над модулями (Module operation), а також права на інсталяцію нового модулю чи його видалення. Дані налаштування задають право користувача на зазначені операції в цілому;

– права доступу до модулів: задаються права доступу до конкретних операцій у БД над конкретними таблицями, а також конкретними модульними операціями конкретного модуля.

Загальні права доступу мають вищий пріоритет, ніж модульні права доступу. Це означає, що у випадку відсутності прав доступу до БД в цілому у користувача, наявність цих прав у конкретному модулі будуть проігноровані.

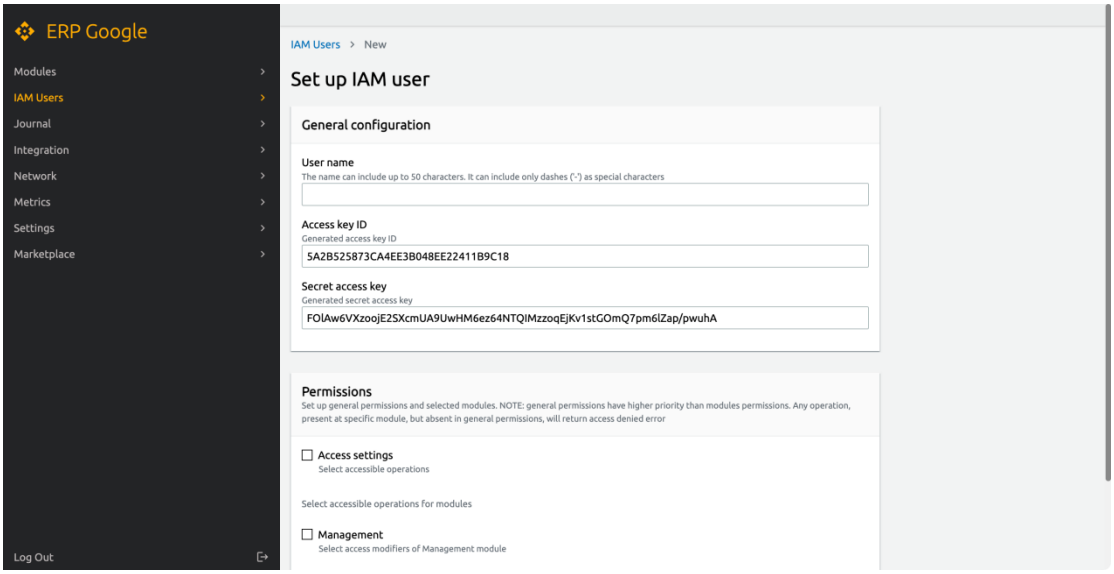

Рисунок 5.45 – Налаштування логіну IAM користувача

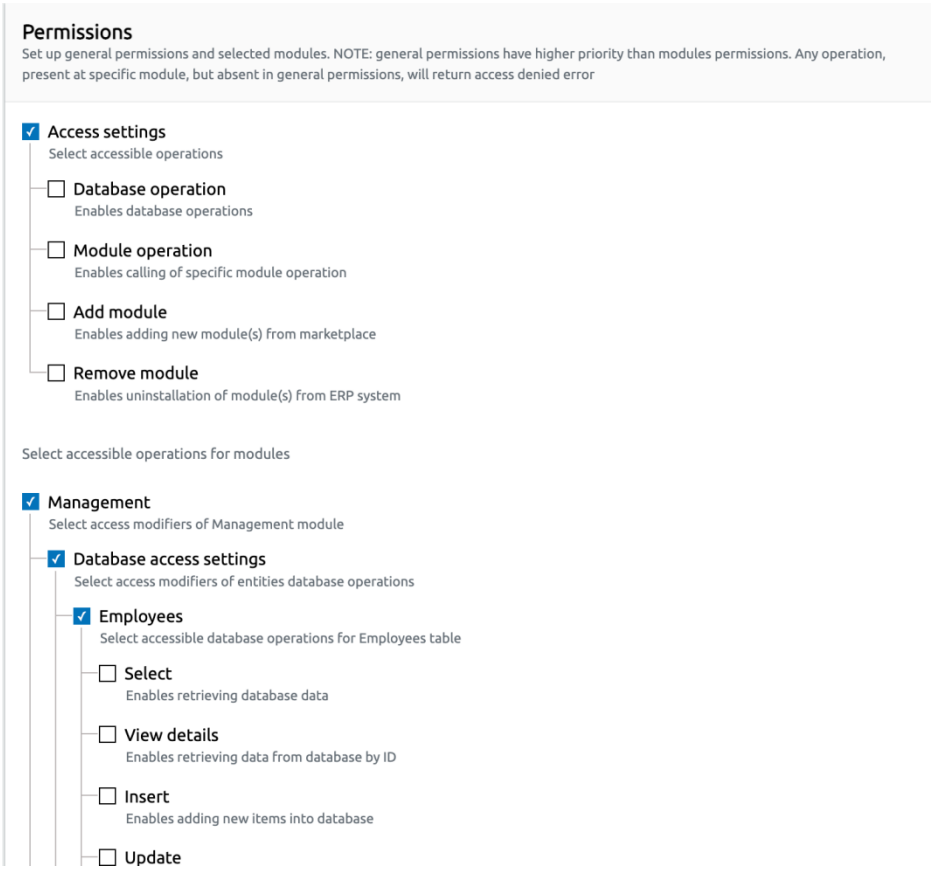

Рисунок 5.46 – Налаштування прав доступу IAM користувача

Механізм інтеграції реалізований схожим чином із IAM адмініструванням (рисунок 5.47): створюється інтеграція із двома можливими видам: ESB або HTTP. ESB інтеграція реалізована у вигляді SignalR хабу, який спрацьовує у момент спрацювання тієї чи іншої події (яка зазначається при налаштувань прав доступу у інтеграції). HTTP інтеграція реалізована у вигляді звичайного HTTP запиту: при створенні зазначається URL, на який відбувається запит із вказаним типом та заголовками.

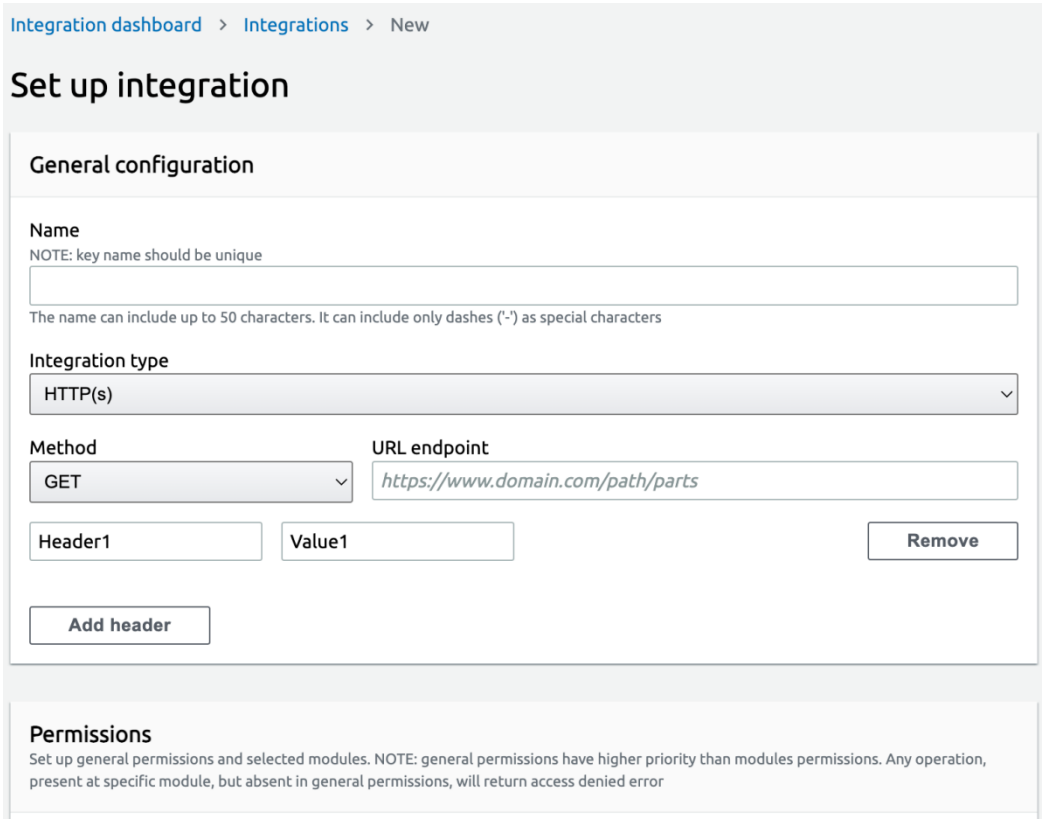

Рисунок 5.47 – Приклад створення HTTP інтеграції

5.11 Розробка консолі взаємодії

Консоль управління є веб-застосунком, аналогічно опублікований у хмарі та виступає в якості зручного інструменту роботи із ERP системами. Звісно, кожна ERP система приймає запити не лише із консолі управління, а будь-які HTTP запити, проте консоль управління є зручним інструментом із веб-інтерфейсом.

Консоль управління захищена авторизацією аналогічним чином як і веб-сервіс ERP системи: при спробі перейти на будь-яку сторінку неавторизованому користувачеві його буду перенаправлено на сторінку авторизації (рисунок 5.48).

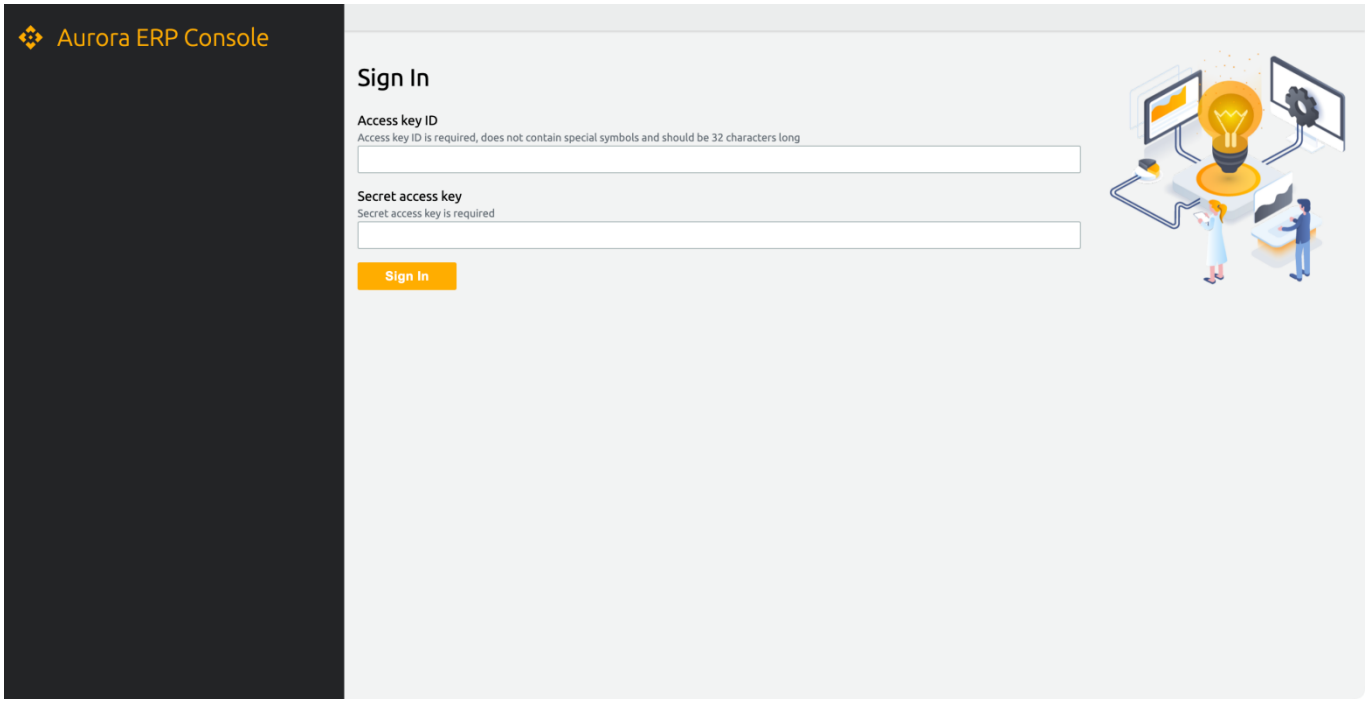

Рисунок 5.48 – Сторінка авторизації у консолі управління

На сторінці авторизації користувач має ввести ключ доступу та секретний ключ IAM користувача, створеного у ERP системі (рисунок 5.45). Завдяки цим даним консоль управління ідентифікує, до якої саме системи потрібно робити запит. У випадку успішної авторизації користувач зможе повністю працювати із системою, в залежності від тих прав, що у нього будуть в наявності, наприклад, переглядати дані таблиць модулів (рисунок 5.49).

Аналогічно з веб-сервісом сесія користувача може бути активною упродовж 12 годин. Завдяки цьому користувачеві не доведеться кожного разу переавторизовуватися при новому відкритті веб сторінки у браузері.

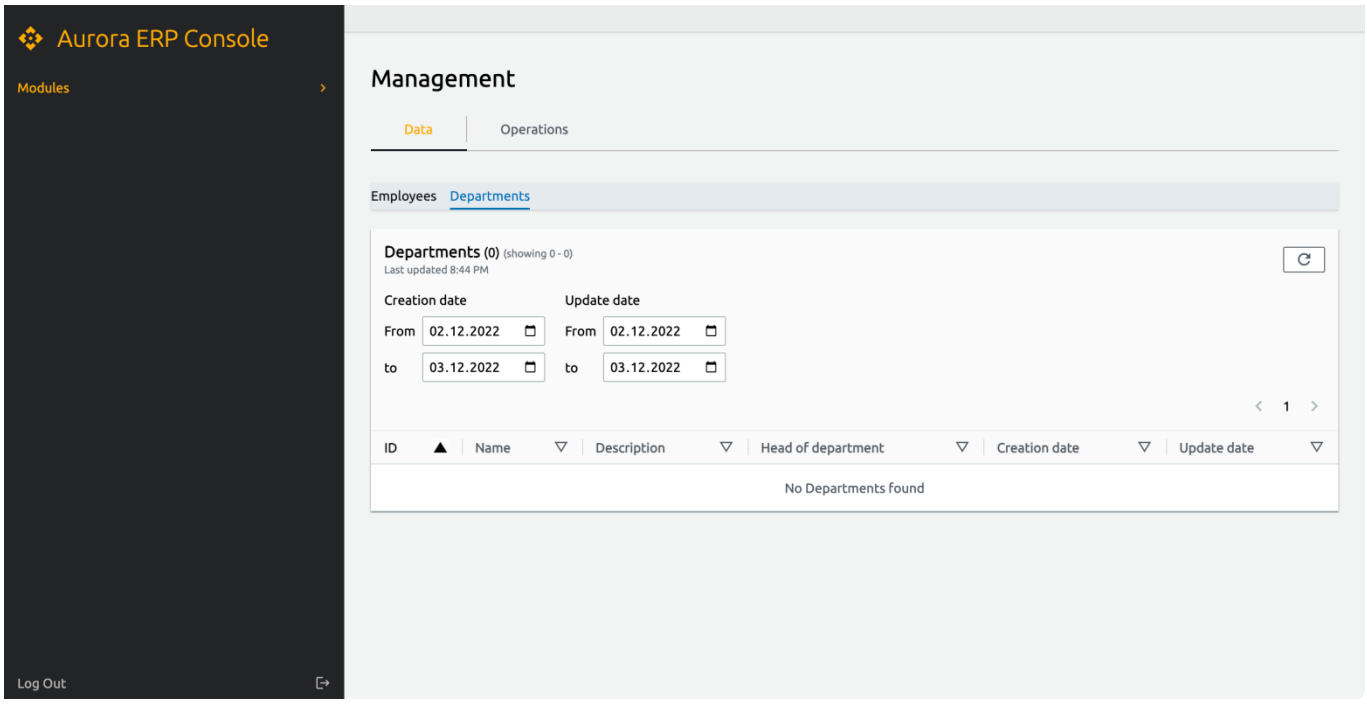

Рисунок 5.49 – Перегляд даних ERP систем через консоль управління

Висновок до розділу 5

У даному розділі був розроблений фреймворк із створення веб-сервісів управління ресурсами підприємства, починаючи з розробки загальної структури системи, переліку застосунків, завершуючи реалізацією конкретних функціональних можливостей. В кожному підрозділі даного розділу описується процес розробки кожного застосунку-компонента фреймворку, технології та архітектурні принципи та підходи, що були використані при його розробці, його взаємодія із іншими компонентами фреймворку.

Окремої уваги заслуговує публікація застосунків у хмарі: були описані та аргументовані причини використання того чи іншого хмарного сервісу, що був використаний при розробці того чи іншого застосунку.

Підходи та архітектурні рішення, що були використанні в процесі розробки, використовуються у більшості сучасних застосунків найбільших IT гігантів світу, тому в їх надійності та коректності немає сумнівів.

6 РОЗРОБКА СТАРТАП-ПРОЄКТУ

Метою даного розділу є маркетингове дослідження та аналіз стартап-проєкту, що характеризуватиме можливість впровадження цього проєкту на ринку, а також напрямки можливих реалізацій даного впровадження.

6.1 Опис ідеї стартапу

Першим кроком є опис ідеї стартап-проекту, що дасть цілісне уявлення про його зміст та перспективні ринки, де варто шукати потенційних клієнтів. Ідея стартап-проєкту описана у таблиці 6.1.

Таблиця 6.1 – Опис ідеї стартап-проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Фреймворк для швидкого запуску веб-сервісів для управління ресурсами підприємства на основі вибраних користувачем модулів	1. Малий бізнес	Створення інформаційної системи з нуля без необхідності замовлення її розробки
	2. Середній бізнес	Доповнити можливої існуючої системи новим функціоналом шляхом інтеграції із ERP системою
	3. Великий бізнес	Створення системи для кожного нового застосунку або інтеграція усіх існуючих додатків в єдину систему

Таблиця 6.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проєкту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Microsoft Dynamics 365	Oracle ERP			
1.	Необхідність залучення команди розробників	-	+	+			+
2.	Простота розгортання	+	-	-			+
3.	Різноманіття фунціоналу	-	+	+	-		
4.	Зручність інтерфейсу	+	-	+		-	
5.	Налаштування прав доступу	+	+	+		-	
6.	Універсальність запитів до системи	+	-	-			+
7.	Інтеграція із зовнішніми системами	+	+	+		-	
8.	Наявність SDK	+	+	+		-	
9.	Наявність мобільного застосунку	-	+	+	-		
10.	Простота контент менеджменту	-	+	+	-		

6.2 Технологічний аудит ідеї проєкту

В даному підрозділі було розглянуто технологічну здійсненність ідеї проекту. Дані цього аудиту наведені у таблиці 6.3.

Таблиця 6.3 – Технологічна здійсненність ідеї проєкту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1.	Інтеграція системи із сторонніми додатками	SignalR hub, HttpClient	Наявна	Доступна безкоштовно
2.	Підпис запитів	HMAC SHA256	Наявна	Доступна безкоштовно
3.	Реалізація прав доступу	Власна реалізація	Наявна	Доступна безкоштовно
4.	Підтримка декількох видів БД	Власна реалізація	Наявна	Доступна безкоштовно
5.	Простота розробки, єдина кодова база для всіх застосунків	C#/.NET, Blazor Server	Наявна	Доступна безкоштовно
6.	Створення мобільного застосунку	C#, Xamarin	Наявна	Доступна безкоштовно
7.	Публікація застосунків у хмарному середовищі	AWS	Наявна	Доступна безкоштовно впродовж 1 року або/та для малих потужностей, інакше – платно
Обрана технологія реалізації ідеї проекту: є цілком можливою				

6.3 Аналіз ринкових можливостей запуску стартап-проєкту

Наступним кроком є визначення можливостей, що присутні на ринку, аби використати їх під час впровадження проєкту. Окрім цього визначені ринкові загрози, які можуть завадити реалізації ідеї проєкту. Цей аналіз наведений у таблицях 6.4 і 6.5.

Таблиця 6.4 – Попередня характеристика потенційного ринку

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	2 (розробники, бізнес)
2	Загальний обсяг продаж, грн/ум.од	Вимірюється мільярдами доларів на рік, точна сума невідома через корпоративну таємницю
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Для повної реалізації проєкту необхідно лише юридично зареєструвати підприємницьку діяльність
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	53%

Таблиця 6.5 – Характеристика потенційних клієнтів

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1.	Швидке і просте впровадження ERP системи	Власники бізнесу	Створення інформаційної системи з нуля без необхідності замовлення її розробки	Простота та швидкість впровадження. Доступні ціни
2.	Простота роботи із корпоративними даними підприємства	Працівники підприємств	Можливість швидкої авторизації у системі та простій роботі із даними	Інтуїтивно зрозумілий інтерфейс. Набір необхідних функцій.

6.4 Розробка ринкової стратегії проєкту

Далі був проведений аналіз ринку на предмет факторів ризику та можливостей (таблиці 6.6 – 6.7) та сприяння розвитку проєкту: ступеневий, конкуренції, аналіз слабких та сильних сторін (таблиці 6.8 – 6.13).

Таблиця 6.6. – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Подорожчання	Збільшення цін на хмарні сервіси	Повідомити користувачів про підвищення цін та обґрунтувати це явище
2.	Конкуренція	Поява конкурентів, що надають схожі сервіси.	Розробка нового функціоналу, покращення технічних характеристик сервісу
3.	Зменшення фінансування	Недостатнє фінансування у довготривалій перспективі.	Залучення нових інвестицій. Постійне оновлення та покращення сервісу, збільшення ефективності його роботи
4.	Збільшення навантаження на систему	При рості кількості користувачів збільшиться і навантаження на застосунки	Оновити архітектуру використання хмарних сервісів. Використовувати дорожчі, але ефективніші сервери

Таблиця 6.7. – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1.	Новий модуль	Збільшення різноманіття функціоналу, що надається користувачам	Розробка нового функціоналу

2.	Реклама	Можливість рекламної інтеграції із іншими компаніями	Виділення додаткових коштів на рекламну компанію
3.	Збільшення кількості клієнтів	При покращенні якості сервісу, користувачі з великою вірогідністю продовжуватимуть його використовувати.	Покращення існуючого функціоналу та впровадження нового. Заохочення нових клієнтів.

Таблиця 6.8. – Ступеневий аналіз конкуренції ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції: чиста	Існують досконалі аналоги з повним або частковою підтримкою функціоналу	Покращення функціоналу та його урізноманітнення
2. За рівнем конкурентної боротьби: міжнародний	Існуючі аналоги розроблялися різними країнами	Створення локалізацій системи
3. За галузевою ознакою: внутрішньогалузева	Даний ПЗ може використовуватися лише в галузі ІТ	Впровадження нового функціоналу
4. Конкуренція за видами товарів: товарно-родова	Конкурують товари одного виду	Розширення функціоналу
5. За характером конкурентних переваг нецінова	Доступні унікальні функції. Більш широкий функціонал	Впровадження більш різноманітного функціоналу

6. За інтенсивністю: марочна	Наявність унікальних ознак, що зробить продукт відомим.	Створення комерційного дизайну

Таблиця 6.9. – Аналіз конкуренції в галузі за М.Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Microsoft Dynamics 365, Oracle ERP	Salesforce, IT Enterprise ERP	Хмарні середовища	Бізнес різних сфер	Існуючі сервіси
Висновки:	Невелика кількість конкурентів, з повною або частковою реалізацією функціоналу.	Потенційних конкурентів достатньо, з серйозними розробками	Залежність невелика, оскільки на ринку хмарних середовищ також присутня конкуренція	Диктують, оскільки функціонал розроблятиметься саме під їх потреби	Клієнти користуються існуючими аналогами, що ускладнить заохочення користувачів

Таблиця 6.10. – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1.	Зручність використання	Спроєктована система зручна і зрозуміла у використанні
2.	Комплексне рішення	Підвищення ефективності за рахунок поєднання декількох додатків в одній системі
3.	Можливість розширення	Архітектура сервісу дозволяє розширювати систему новим функціоналом
4.	Унікальність функцій	Наявність унікальних для ринку систем функцій
5.	Платформонезалежність	Відсутність вимог до операційної системи

Таблиця 6.11. – Порівняльний аналіз сильних та слабких сторін проєкту

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з проєктом						
			–3	–2	–1	0	+1	+2	+3
1.	Зручність використання	15				MS	Oracle		
2.	Комплексне рішення	15				MS, Oracle			
3.	Можливість розширення	15					Oracle, MS		
4.	Унікальність функцій	15					MS, Oracle		
5.	Платформонезалежність	20				MS, Oracle			

Таблиця 6.12. – SWOT аналіз

Сильні сторони: – Можливість до розширення – Простота використання – Зрозумілий інтерфейс – Унікальний функціонал	Слабкі сторони: – Відсутність аналітики – Мале різноманіття функціоналу
Можливості: – Розробка нових модулів – Покращення інтерфейсу – Оптимізація – Розширення функціоналу	Загрози: – Достатня конкуренція – Недостатнє фінансування

Таблиця 6.13. – Альтернативи ринкового впровадження

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Впровадити безкоштовний пробний період	Велика	2-3 місяці
2.	Презентація нового модулю	Вище середнього	1-2 місяці
3.	Презентація проєкту на конференціях	Велика	2-3 місяці

6.5 Розробка маркетингової програми стартап-проєкту

Останнім кроком с розробка маркетингової програми. В ході розробки стратегії потрібно визначити цільову групу, стратегію позиціонування тощо. Дана розробка представлена у таблицях 6.14 – 6.22.

Таблиця 6.14. – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1.	Власники малого бізнесу	Висока	Високий	Висока	Висока
2.	Власники середнього бізнесу	Середня	Середній	Висока	Середня
3.	Власники великого бізнесу	Низька	Середній	Висока	Середня
Які цільові групи обрано: власники бізнесу різного типу та сфер					

Таблиця 6.15. – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
	Впровадження нового функціоналу	Охоплення нових сфер бізнесу	Збільшення конкурентоспроможності	Стратегія диференціації

Таблиця 6.16. – Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
	Ні, система є зміною існуючих аналогів	Компанія буде як шукати нових споживачів, так і забирати існуючих у конкурентів	Компанія буде копіювати базовий функціонал та полегшувати процес роботи в цілому	Виклик лідера

Таблиця 6.17. – Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1.	Доступність	Зменшення вартості для користувачів	Використання безкоштовних технологій	Дешевизна
2.	Простота використання	Аналіз потреб користувача	Швидкість та простота роботи через зрозумілий інтерфейс	Зрозумілість

3.	Наявність потрібного функціоналу	Формування вимог до розробки	Можливість інтеграції та налаштувань прав доступу	Стабільність

Таблиця 6.18. – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1.	Потреба в швидкому впровадженні системи управління	Система управління ресурсами підприємства з нуля	Швидке та легке впровадження системи управління
2.	Потреба в простій взаємодії із ERP системою	Взаємодія із корпоративними даними через зручний веб-застосунок	Проста і зручна взаємодія
3.	Потреба в безпечному обміні даними між працівникам компанії	Сучасний алгоритм підпису запиту	Надійний механізм безпеки передачі даних

Таблиця 6.19. – Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>I. Товар за задумом</td><td colspan="3">Фреймворк зі створення веб-сервісів управління ресурсами підприємства на платформі .NET з використанням технології Blazor Server</td></tr>
<tr><td rowspan="6">II. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>1. Просте і швидке розгортання ERP системи
2. Відмова від витрат часу та фінансів на розробку системи
3. Розширення функціоналу
4. Безпека даних при передачі</td><td>Нм
Нм
Нм
М</td><td>Вр/Тх
Вр/Тх
Ор/Вр/Тх
Тх</td></tr>
<tr><td colspan="3">Якість: дотримання стандартів розробки архітектури та використання сучасних технологій</td></tr>
<tr><td colspan="3">Сервіс</td></tr>
<tr><td colspan="3">Марка: Aurora ERP</td></tr>
<tr><td rowspan="2">III. Товар із підкріпленням</td><td colspan="3">До продажу: документація, знижки при довгостроковому використанні.</td></tr>
<tr><td colspan="3">Після продажу: технічна підтримка, консультація, впровадження нового функціоналу</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання: реєстрація права на інтелектуальну власність, закритий код продукту.</td></tr>
</table>

Таблиця 6.20. – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
	10000 грн/міс	180000 грн.	10-50 млн євро/рік	100000-500000 грн.

Таблиця 6.21. – Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
	Запуск системи веб-сайт або запит	Автоматизоване розгортання системи у хмарі	Глибока	Реклама

Таблиця 6.22. – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
	Бажання встановити систему управління	Інтернет	Можливість до розширення, унікальні можливості	Висвітлити переваги сервісу, його набір функцій	Акцент на унікальних функціях, простоті використання

Висновок до розділу 6

В даному розділі були проаналізовані можливості розвитку розробленого фреймворку в якості як стартап-проекту. Аналіз показав, що у проєкта є перспектива ринкової комерціалізації через присутній попит, зростаючу динаміка ринку та високу рентабельність. Були визначені потенційні групи користувачів та конкурентоспроможність. Імплементація проєкту є цілком доцільною.

ВИСНОВОК

Результатом магістерської роботи став фреймворк для створення веб-сервісів управління ресурсами підприємства, що повністю відповідає всім технічним та функціональним вимогам. Сервіс був успішно протестований і повністю відповідає поставленим задачам.

В ході виконання магістерської роботи були проаналізовані сучасні ERP системи, і що вони із себе представляють. В рамках аналізу були визначені основні принципи їхньої роботи та архітектури, був проведений огляд систем-аналогів, детально розглянута концепція модульності, аргументовані причини використання даної концепції та її переваги.

На основі проведеного аналізу та огляду був сформований перелік важливих технічних та функціональних вимог до фреймворку. На основі отриманих вимог був зроблений вибір технологій, принципів і шаблонів проєктування та архітектури, які стали в нагоді в процесі розробки та значно полегшили його, а також процес тестування фреймворку.

За допомогою вибраних сучасних стандартів, шаблонів і принципів проєктування і розробки був створений фреймворк із перспективою його подальшого розвитку як комерційного проєкту. Фреймворком зручно користуватись як з точку зору клієнта, так і з точки зору адміністраторів фреймворку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Что такое MRP. URL: https://www.sap.com/cis/insights/what-is-mrp.html (дата звернення: 31.10.2022).

2. Modular ERP: advantages and disadvantages. URL: https://blog.vault-erp.com/post/2021/10/21/modular-erp-advantages-and-disadvantages (дата звернення: 31.10.2022).

3. Top ERP security problems and best practices. URL: https://www.netsuite.com/portal/resource/articles/erp/erp-security-best-practices.shtml (дата звернення: 31.10.2022).

4. Overview of integration with ERP components. URL: https://www.ibm.com/docs/en/order-management-sw/10.0?topic=systems-overview-integration-erp-components (дата звернення: 31.10.2022).

5. ESB. URL: https://habr.com/ru/company/pnn/blog/191600/ (дата звернення: 03.11.2022).

6. Какие задачи решают IAM системы. URL: https://habr.com/ru/post/221159/ (дата звернення: 03.11.2022).

7. Microsoft Dynamics 365. URL: https://dynamics.microsoft.com/en-us/ (дата звернення: 04.11.2022).

8. Oracle NetSuite. URL: https://www.netsuite.com/portal/home.shtml (дата звернення: 04.11.2022).

9. Oracle ERP Cloud. URL: https://en.wikipedia.org/wiki/Oracle_Cloud_Enterprise_Resource_Planning (дата звернення: 04.11.2022).

10. Epicor Kinetic. URL: https://www.epicor.com/en/industry-productivity-solutions/manufacturing/platforms/kinetic/ (дата звернення: 04.11.2022).

11. SAP S/4HANA. URL: https://www.sap.com/ukraine/products/erp/s4hana.html (дата звернення: 04.11.2022).

12. API vs. SDK: The Difference Explained. URL: https://getstream.io/glossary/api-vs-sdk/ (дата звернення: 05.11.2022).

13 .NET Platform. URL: https://dotnet.microsoft.com/ (дата звернення: 07.11.2022).

14. C# Documentation. URL: https://docs.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 07.11.2022).

15. Blazor. URL: https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor (дата звернення: 07.11.2022).

16. SignalR Core. URL: https://habr.com/ru/company/jugru/blog/349096/ (дата звернення: 07.11.2022).

17. Entity Framework Core. URL: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx (дата звернення: 07.112022).

18. CQRS Pattern. URL: https://habr.com/ru/post/347908/ (дата звернення: 07.11.2022).

19. Зачем нужен MediatR. URL: https://habr.com/ru/post/588887/ (дата звернення: 07.11.2022).

20. СУБД. URL: https://itglobal.com/ruru/company/glossary/subd-sistema-upravleniya-bazami-dannyh/ (дата звернення: 07.11.2022).

21. PostgreSQL. URL: https://www.postgresql.org/ (дата звернення: 07.11.2022).

22. Amazon Web Services. URL: https://aws.amazon.com (дата звернення: 10.11.2022).

23. Google Cloud Platform. URL: https://cloud.google.com (дата звернення: 10.11.2022).

24. Microsoft Azure. URL: https://azure.microsoft.com/en-us/ (дата звернення: 10.11.2022).

25. Amazon ARN. URL: https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html (дата звернення: 11.11.2022).

26. RFC 4868 HMAC SHA256. URL: https://www.rfc-editor.org/rfc/rfc4868 (дата звернення: 11.11.2022).

27. Canonical requests. URL: https://cloud.google.com/storage/docs/authentication/canonical-requests (дата звернення: 12.11.2022).