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

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2022 р.

Дипломний проєкт

на здобуття ступеня бакалавра

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

спеціальності «121 Інженерія програмного забезпечення»

на тему: Програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою

Виконав студент IV курсу, групи ІП-81
(шифр групи)
Хмара Владислав Віталійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник ст.викладач, Головченко М.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації ст.викладач, Головченко М.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент каф. ІСТ, к.т.н., доц., Жураковська О. С. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ –2022

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

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення комп'ютеризованих систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2022 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Хмарі Владиславу Віталійовичу
(прізвище, ім'я, по батькові)

Тема проєкту — Програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою

керівник проєкту — Головченко Максим Миколайович, ст.викладач
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «10» червня 2022 р. №1033-с

2. Термін подання студентом проєкту « 19 » червня 2022 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки н
1) Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі.
2) Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних.
3) Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв'язання.
4) Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів.
5) Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань _____

2) Схема структурна компонентів _____

3) Схема бази даних _____

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «10» березня 2022 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення літератури за тематикою проекту	10.03.2022	
2	Аналіз існуючих методів розв’язання задачі	17.03.2022	
3	Постановка та формалізація задачі	17.03.2022	
4	Проектування структури програмного забезпечення	31.03.2022	
5	Обґрунтування вибору використаних технічних засобів	31.03.2022	
6	Розробка програмного забезпечення	24.04.2022	
7	Тестування програмного забезпечення	01.05.2022	
8	Виконання графічних документів	22.05.2022	
9	Оформлення пояснювальної записки	22.05.2022	
10	Подання ДП на попередній захист	06.06.2022	
11	Подання ДП рецензенту	12.06.2022	
12	Подання ДП на основний захист	19.06.2022	

Студент _____ Владислав ХМАРА

(підпис) (ініціали, прізвище)

Керівник _____ Максим ГОЛОВЧЕНКО

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 95 таблиць, 22 рисунків та 13 джерел – загалом 96 сторінок.

Дипломний проєкт присвячений розробці програмного забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою.

Метою розробки є спрощення розробки веб-застосунків з мікросервісною архітектурою шляхом надання графічного інтерфейсу для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою, що включає у себе:

– генерацію конфігураційних файлів для системи оркестрації контейнерів kubernetes для розгортання мікросервісів, баз даних, брокерів подій тощо;

– генерацію конфігураційних файлів для систем CI/CD;

– генерацію частини коду мікросервісів.

У розділі першому досліджена предметна область, розглянуті технічні рішення і аналоги, розроблені вимоги.

Розділ другий присвячений моделюванню бізнес-процесів, створенню архітектури і конструюванню програмного забезпечення.

У третьому розділі розглядається аналіз якості програмного забезпечення та описуються процеси тестування.

У четвертому розділі розглядається розгортання і випуск програмного забезпечення.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, КОНСОЛЬНИЙ ДОДАТОК, МІКРОСЕРВІСНА АРХІТЕКТУРА, DOCKER, KUBERNETES, KAFKA, БАЗА ДАНИХ, МІКРОСЕРВІСИ.

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 95 tables, 24 figures and 13 sources - a total of 94 pages.

The diploma project is devoted to the development of software for automated design of web applications with microservice architecture.

The purpose of the development is to simplify the development of web applications with microservice architecture by providing a graphical interface for automated design of web applications with microservice architecture, which includes:

– generation of kubernetes configuration files for the deployment of microservices, databases, event brokers, etc.;

– generation of configuration files for CI / CD systems;

– generation of part of the microservice code.

In the first section the subject area is investigated, technical solutions and analogues are considered, requirements are developed.

The second section is devoted to business process modeling, architecture creation and software design.

The third section discusses software quality analysis and describes testing processes.

The fourth section discusses software deployment and release.

KEY WORDS: WEB APPLICATION, CONSOLE APPLICATION, MICROSERVATION ARCHITECTURE, DOCKER, KUBERNETES, KAFKA, DATABASE, MICROSERVICES.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-8321.045490.01.91	Технічне завдання	12	
3	А4	КПІ.ІП-8321.045490.02.81	Пояснювальна записка	94	
4	А4	КПІ.ІП-8321.045490.03.13	Опис програми	37	
5	А4	КПІ.ІП-8321.045490.04.51	Програма та методика тестування	7	
6	А4	КПІ.ІП-8321.045490.05.34	Керівництво користувача	11	
7	А3	КПІ.ІП-8321.045490.06.99	Схема структурна варіантів	1	
			використань		
8	А3	КПІ.ІП-8321.045490.07.99	Схема бази даних	1	
9	А3	КПІ.ІП-8321.045490.08.99	Креслення вигляду екранних форм	1	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП–8321.045490.00.90		
Розроб.		Хмара В.В.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Головченко М.М.					
Н.контр.		Головченко М.М.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП–81	
Затв.		Жариков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2022 р.

Програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою

Технічне завдання

КПІ.ІП-8321.045490.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Максим ГОЛОВЧЕНКО

Нормоконтроль:

_____ Максим ГОЛОВЧЕНКО

Виконавець:

_____ Владислав ХМАРА

Київ – 2022

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою.

Галузь застосування:

Наведене технічне завдання поширюється на розробку програмного забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою. Програмне забезпечення призначене для веб-розробників і DevOps інженерів, які розробляють веб-застосунки з використанням мікросервісної архітектури.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки програмного забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для веб-розробників і DevOps інженерів, які розробляють веб-застосунки з використанням мікросервісної архітектури.

Метою розробки є спрощення розробки веб-застосунків з мікросервісною архітектурою, надання графічного інтерфейсу для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою, що включає у себе:

– генерацію конфігураційних файлів для системи оркестрації контейнерів kubernetes для розгортання мікросервісів, баз даних, брокерів подій тощо;

– генерацію конфігураційних файлів для систем безперервної інтеграції і доставки;

– генерацію частини коду мікросервісів.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

4.1.1.1 Для користувача:

- реєстрація користувача за допомогою пошти і пароля;
    - перевірка електронної пошти на відповідність формату електронної пошти;
    - перевірка електронної пошти на унікальність;
    - перевірка паролю на відповідність мінімальній і максимальній довжині;
    - перевірка паролю на наявність необхідних символів (хоча б одна англійська літера, одне число та один спеціальний символ);
- авторизація користувача за допомогою пошти і паролю;
    - перевірка пошти і паролю на відповідність;
- створення нового проекту;
- збереження проекту;
    - перевірка проекту на наявність помилок;
- завантаження збереженого проекту;
- видалення проекту;
- додавання мікросервісів за допомогою графічного інтерфейсу;
- додавання баз даних за допомогою графічного інтерфейсу;
- додавання API шлюзів за допомогою графічного інтерфейсу;
- додавання брокерів повідомлень за допомогою графічного інтерфейсу;
- налаштування мікросервісів;
    - введення назви мікросервісу;

- введення порту, за яким можна отримати доступ до мікросервісу;
- введення кількості реплік для мікросервісу;
- вибір типу мікросервісу;

– налаштування баз даних;
- введення назви бази даних;
- введення розміру сховища для бази даних;
- вибір типу бази даних;

– налаштування API шлюзів;
- введення назви хоста;

– налаштування брокерів повідомлень;
- введення кількості реплік;
- вибір типу брокера повідомлень;

– видалення елементів: мікросервісів, баз даних, API шлюзів, брокерів повідомлень;
– створення зв’язків між сумісними елементами;
– експорт проекту у вигляді конфігураційного файлу для консольного застосунку, який виконує генерацію макету веб-додатку;
– генерація конфігураційних файлів kubernetes для розгортання мікросервісів;
– генерація конфігураційних файлів kubernetes для розгортання баз даних;
– генерація конфігураційних файлів kubernetes для розгортання API шлюзу;
– генерація конфігураційних файлів kubernetes для розгортання брокерів подій;
– генерація команд для створення хмарної інфраструктури;
– генерація макету мікросервісу на базі опису API;
– генерація документації з описом API на базі OpenAPI.

4.1.2 Розробку виконати мовою Typescript на платформі NodeJs, для клієнтської частини веб-додатку використати фреймворк Angular, для серверної частини використати фреймворк NestJs

4.1.3 Додаткові вимоги:

– Зрозумілий інтерфейс користувача для веб-додатку;
– Наявність довідки для консольного додатку.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

4.2.2 Передбачити захист від некоректних дій користувача.

4.2.3 Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

4.4.2 Мінімальна конфігурація технічних засобів:

4.4.2.1 Тип процесору ..... Intel Core i5.

4.4.2.2 Об‘єм ОЗП..... 4 Гб.

4.4.2.3 Підключення до мережі Інтернет зі швидкістю від 20 мегабіт

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1 Веб-додаток повинен працювати під управлінням усіх операційних систем які підтримують підключення до мережі інтернет та

можуть надати доступ до веб-додатку через браузер, консольний додаток повинен працювати під управлінням операційних систем сімейства Linux.

4.5.2 Вхідні дані вводяться користувачем за допомогою веб-додатку.

4.5.3 Результати повинні бути представлені в наступному форматі: набір файлів, в тому числі файли конфігурації yaml та json, файли з текстом програм ts, js, sh тощо.

4.5.4 Консольний додаток повинен отримувати інформацію про додаток, макет котрого необхідно згенерувати за допомогою конфігураційного файлу типу json.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Згенерувати установчу версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

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

5.2 Програмне забезпечення повинно мати довідникову систему.

5.3 У склад супроводжувальної документації повинні входити наступні документи:

5.3.1 Пояснювальна записка не менше ніж на 90 аркушах формату А4 (без додатків 5.3.2 - 5.3.6).

5.3.2 Технічне завдання.

5.3.3 Керівництво користувача.

5.3.4 Програма та методика тестування

5.4 Графічна частина повинна бути виконана на 3 аркушах формату А3, котрі включаються у якості додатків до пояснювальної записки:

5.4.1 Схема структурна варіантів використання.

5.4.2 Схема бази даних.

5.4.3 Креслення вигляду екранних форм.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	10.03.2022	
2.	Аналіз існуючих методів розв’язання задачі	17.03.2022	
3.	Постановка та формалізація задачі	17.03.2022	Технічне завдання
4.	Проектування структури програмного забезпечення	31.03.2022	Специфікація компонентів
5.	Обґрунтування вибору використаних технічних засобів	31.03.2022	
6.	Розробка програмного забезпечення	24.04.2022	Тексти програмного забезпечення
7.	Тестування програмного забезпечення	01.05.2022	Тести, результати тестування
8.	Виконання графічних документів	22.05.2022	Пояснювальна записка
9.	Оформлення пояснювальної записки	22.05.2022	Графічний матеріал проекту
10.	Подання ДП на попередній захист	06.06.2022	
11.	Подання ДП рецензенту	12.06.2022	
12.	Подання ДП на основний захист	19.06.2022	

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

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

Пояснювальна записка
до дипломного проєкту

на тему: Програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою

КПІ.ІП-8321.045490.02.81

Київ – 2022

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний Інтерфейс
CI/CD	–	Continuous Integration/Continuous Delivery – Безперервна інтеграція і розгортання
BPMN	–	Business Process Model and Notation – Модель і нотація бізнес-процесів
ОС	–	Операційна система.
БД	–	База даних.

ВСТУП

Зараз однією з найбільш популярних архітектур є мікросервісна архітектура. Цією архітектурою користуються Uber, Netflix, Amazon, Ebay, SoundCloud, Zalando і ще багато компаній.

Вона виникла для того, щоб вирішити проблеми традиційної монолітної архітектури, які особливо часто виникають у enterprise розробці. Такими проблемами є:

– висока зв’язність;
– складність CI/CD;
– складність розгортання;
– складність масштабування.

Популяризації мікросервісної архітектури передували деякі фактори, які зробили її можливою. Одним з таких факторів стало підвищення доступності контейнерів і систем оркестрації контейнерів. Так Docker з’явився у 2013 році, а Kubernetes з’явився у 2014.

Перед цим для розгортання мікросервісів використовувались віртуальні машини, що було дуже неефективно через те, що кожна віртуальна машина має свою гостьову операційну систему.

Іншими факторами були доступність комунікаційної інфраструктури, збільшення доступності інструментів для розробки мікросервісів, накопичення знань про мікросервіси, формування паттернів тощо.

Таким чином мікросервісна архітектура прийшла до популярності, проте використання мікросервісів призвело і до проблем. Однією з таких проблем є висока складність управління великою кількістю сервісів, що призводить до ускладнення інфраструктури. Крім того при написанні сервісів часто доводиться повторювати те, що для монолітного застосунку треба було додавати лише один раз. Так наприклад кожен сервіс повинен мати свою окрему базу даних. Це значно сповільнює розробку, а також підвищує ризик виникнення помилок.

Автоматизація дозволяє знизити важкість цих проблем, але сучасні компанії як правило використовують внутрішні інструменти, які не є доступними поза компанією, що пояснює актуальність обраної теми.

Ціллю розробки є спрощення розробки веб-застосунків з мікросервісною архітектурою шляхом надання графічного інтерфейсу для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою, що включає у себе:

– генерацію конфігураційних файлів для системи оркестрації контейнерів kubernetes для розгортання мікросервісів, баз даних, брокерів подій тощо;

– генерацію конфігураційних файлів для систем CI/CD;

– генерацію частини коду мікросервісів.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Архітектура програмного забезпечення – це структура програмної системи, що включає у себе елементи програмного забезпечення, видимі ззовні властивості цих елементів і зв’язки між ними [1]. Наявність формальної архітектури має такі переваги:

– задає чітке бачення проекту для всієї команди;

– надає надійну основу для продукту;

– забезпечує узгодженість підходів і стандартів, що покращує структурованість коду;

– спрощує підтримку коду;

– задає рамки для виявлення і пом’якшення ризиків.

Прикладами архітектури є [2]:

– Багатошарова архітектура – це одна з найпопулярніших архітектур програмного забезпечення. В ній компоненти організовані у горизонтальні шари, кожен з яких виконує певну роль у системі. Як правило цих шарів 3.

– Багаторівнева архітектура – часто така назва використовується взаємозамінно з багатошаровою архітектурою, проте різниця існує. Шар – це логічний структуруючий механізм для елементів програмного рішення, а рівень – це фізичний структуруючий механізм для інфраструктури системи.

– Подійно-орієнтована архітектура – популярна асинхронна розподілена архітектура, що складається зі слабко зв’язаних компонентів, які асинхронно отримують і обробляють події.

– Мікроядерна архітектура – архітектура, що складається з двох типів елементів: ядро системи і плагіни (модулі, що підключаються до ядра).

Мікросервісна архітектура виникла для того, щоб вирішити ряд проблем які виникали внаслідок використання традиційної багаторівневої архітектури. Цими проблемами є[3]:

– висока зв’язність, що призводить до проблем з залежностями між модулями;

– складність розгортання – розгортання в традиційній монолітній архітектурі потребує координації багатьох команд, що сильно ускладнює цей процес;

– складність CI/CD – навіть для найменших змін у програмі розгортання є дуже довгим;

– ускладнена масштабованість – у монолітній архітектурі можна вводити масштабування лише на рівні цілого застосунку, а не, наприклад, конкретного модуля;

Таким чином для вирішення проблем монолітної архітектури мікросервісна архітектура розбиває моноліт на певну кількість модулів, які називають мікросервісами, і які мають одну конкретну ціль, є самодостатніми і незалежними один від одного, та здатні до незалежного розгортання.

Згідно з назвою мікросервіси повинні бути малими, проте ідеальний розмір мікросервісу залежить від багатьох факторів і для різних застосунків буде різним. Так якщо мікросервіс буде занадто великим, то це збільшить складність з якою доведеться справлятись команді що над ним працює, або ж доведеться збільшувати команду. З іншого боку якщо мікросервіс буде дуже малим, то це збільшить витрати на комунікацію між сервісами і зменшить продуктивність застосунку, також це збільшить вимоги до інфраструктури. [4]

Мікросервісна архітектура має ряд переваг (таблиця 1.1) і недоліків (таблиця 1.2). Дана розробка може зменшити тяжкість деяких з недоліків, які має мікросервісна архітектура. Серед них: сильніші вимоги до документації, високі початкові витрати, висока складність DevOps.

Таблиця 1.1 – Переваги мікросервісної архітектури

Назва	Опис
Децентралізація	В мікросервісній архітектурі розробка і розгортання сервісу потребує мінімальної комунікації між командами розробників, що пришвидшує розробку.
Покращення CI/CD	Для мікросервісів розгортання виконується окремо від інших мікросервісів. Це значно зменшує час розгортання, а також знижує пов’язаний з ним ризик.
Висока масштабованість	В мікросервісній архітектурі можна окремо масштабувати кожний сервіс, шляхом кількості запущених екземплярів кожного мікросервісу, що знижує ціну масштабованості.
Підтримка різноманіття технологій всередині одного проекту	На відміну від монолітної архітектури, мікросервіси не обов’язково повинні всі використовувати одні і й ті ж технології. Так це дозволяє використовувати такі технології, які найкраще підходять для вирішення проблем специфічних для сервісу. Також це дозволяє використовувати нові технології без високого ризику, так як проблеми пов’язані з новою технологією будуть обмежені лише тими мікросервісами де вона використовується.
Надійність	У випадку падіння одного мікросервісу проблема обмежується лише ним, також можуть використовуватись працюючі екземпляри мікросервісу або ж екземпляр мікросервісу може бути перезапущений.

Таблиця 1.2 – Недоліки і виклики мікросервісної архітектури

Назва	Опис
Зниження продуктивності	В мікросервісній архітектурі комунікація між сервісами відбувається за допомогою викликів через мережу, що менш ефективно ніж виклики через пам’ять. Також це збільшує вимоги до інфраструктури.
Ненадійна комунікація	Так як мікросервісній архітектурі комунікація між сервісами відбувається за допомогою викликів через мережу, вона є ненадійною. Таким чином при розробці інших мікросервісів потрібно приймати відповідні кроки, щоб забезпечити те, що падіння одного мікросервісу чи проблеми з мережею на призведуть до падіння всієї системи.
Ускладнений рефакторинг	Рефакторинг сильно ускладнюється, у випадку коли певну функціональність необхідно перенести між сервісами, особливо у випадку коли різні сервіси використовують різні технології.
Збільшений ризик вразливостей	Використання різних технологій для сервісів збільшує вразливість системи.
Ускладнене управління базами даних	У випадку якщо, кожен сервіс має свою власну базу даних, значно ускладнюється потік даних, також ускладнюється проведення транзакцій.
Сильніші вимоги до документації	Для забезпечення комунікації між сервісами кожен сервіс повинен мати гарну документацію.
Високі початкові витрати	Перед тим як застосунок може почати працювати, уся відповідна інфраструктура, що включає у себе CI-CD, засоби оркестрації контейнерів тощо повинна бути налаштована.

Продовження таблиці 1.2

Назва	Опис
Велика складність DevOps	Оркестрація та розгортання мікросервісів є значно складнішими ніж розгортання додатку з традиційною монолітною архітектурою

1.2 Змістовний опис і аналіз предметної області

Так як дана розробка передбачає генерацію конфігураційних файлів для налаштування інфраструктури для застосунків з мікросервісною архітектурою, то необхідно проаналізувати складові частини таких застосунків.

Доступність контейнерів була однією з речей, які уможливили мікросервісну архітектуру [3]. З поширенням хмарних технологій, для розгортання мікросервісів почали використовувались віртуальні машини, але вони погано підходять для цієї задачі, адже кожна віртуальна машина містить гостьову операційну систему (рисунок 1.1), яка є дуже важкою.

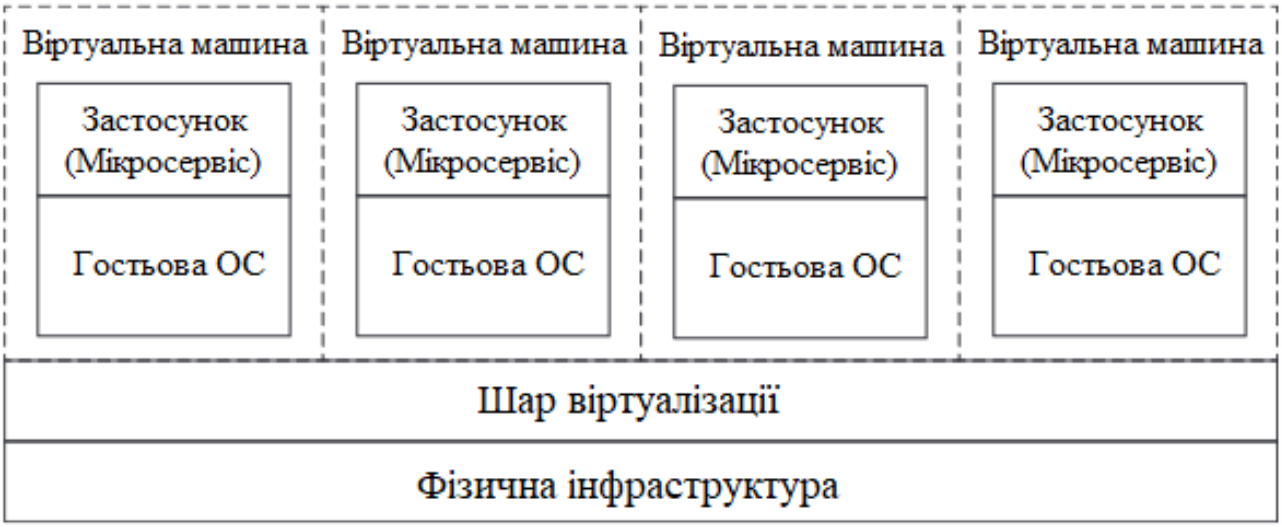


Рисунок 1.1 – Схема віртуалізації

На відміну від віртуальних машин контейнери ізолюють виконання на рівні операційної системи (рисунок 1.2). Таким чином вони використовують

значно менше ресурсів. Майже завжди для кожного мікросервісу використовується свій контейнер.

Рисунок 1.2 – Схема контейнеризації

Контейнери є важливими для автоматизації розгортання. Замість того, щоб оновлювати програмне забезпечення на сервері, сервер повністю розгортається наново. Головною перевагою цього підходу є те, що чисте встановлення значно простіше відтворити ніж оновлення, так як оновлення з дуже різних станів повинно приводити сервер в один стан. У контейнері також знаходяться необхідні залежності, що дозволяє надійно розгортати додаток будь-де без ризиків пов’язаних із сумісністю [5].

Найбільш популярним інструментом для створення і управління контейнерами є Docker [6]. Docker надає високорівневе API, що дозволяє взаємодіяти з легковісними контейнерами Linux. Тому було вирішено підтримувати генерацію конфігураційних файлів Docker у розробці.

Задача виявлення сервісів полягає у тому, що певна система повинна знати про всі розгорнуті контейнери і відповідні їм пари ip-адрес і портів у кожен момент часу, щоб забезпечити маршрутизацію.

Інструменти для вирішення задачі виявлення сервісів існують у спектрі від низькорівенвих рішень, таких як Etcd чи Consul, до високорівневих систем оркестрації контейнерів, таких як Kubernetes і Docker Swarm [5]. Останні

характеризуються високим рівнем автоматизації і абстракції. Так замість того, щоб обирати конкретно які сервіси будуть запускатись на яких серверах, розробник вказує, яка частина ресурсів повинна виділятись на даний сервіс і система оркестрації контейнерів сама балансує їх на основі цих критеріїв.

З них Kubernetes є одним з найбільш популярних. Він надає такі переваги [7]:

– забезпечення відмовостійкості, завдяки тому що у випадку падіння pod інший екземпляр запускається автоматично;

– забезпечення горизонтальної масштабованість за допомогою контролювання кількості реплік кожного pod;

– забезпечення вищої ефективності у використанні ресурсів;

– розділення відповідальності – команда розробки сервісу не повинна взаємодіяти з командою, що займається інфраструктурою;

– моніторинг – kubernetes має вбудовану систему моніторингу ресурсів контейнеру.

Тому було вирішено підтримувати генерацію конфігураційних файлів Kubernetes у розробці.

Існують два основних види комунікації між сервісами синхронний і асинхронний. У випадку синхронної передачі відправник відправляє запит до отримувача і блокується поки не отримає відповідь (рисунок 2.3). Прикладом такої комунікації є прямий виклик REST API. При асинхронній комунікації запит і відповідь відбуваються незалежно один від одного (рисунок 2.4). Прикладом технології для асинхронної комунікації є брокери повідомлень, такі як Kafka чи RabbitMQ. Також існує гібридна комунікація. Вона полягає в тому, що сервіс підтримує обидва види комунікації, наприклад, комунікацію по HTTP і передачу повідомлень за допомогою Kafka. Основною перевагою такого підходу є гнучкість.

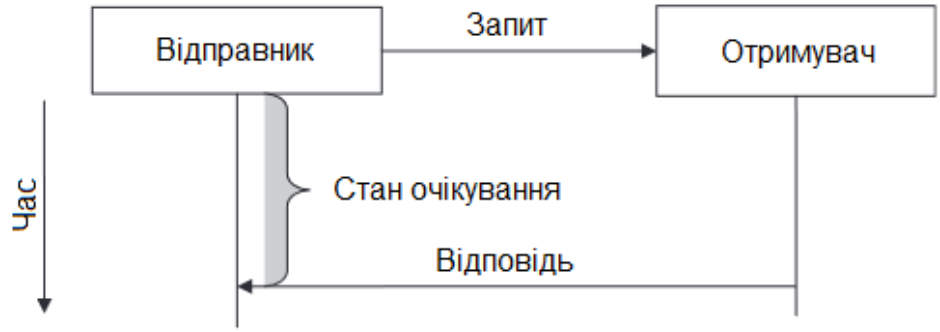


Рисунок 1.3 – Синхронна комунікація

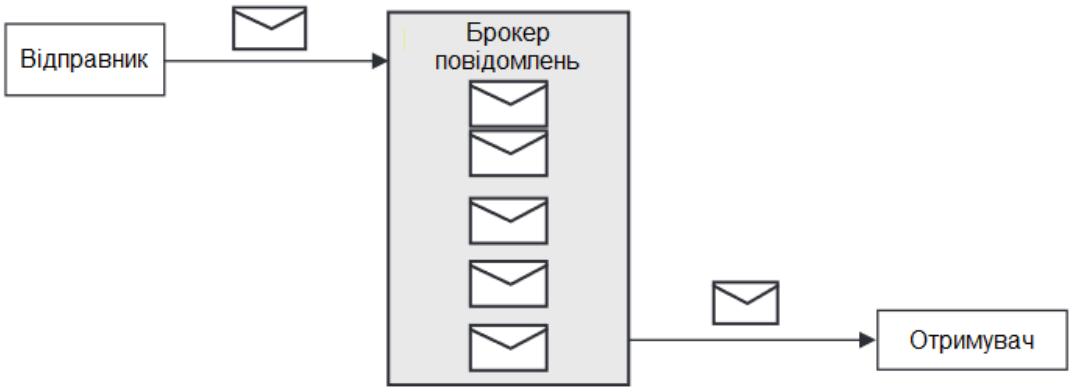


Рисунок 1.4 – Асинхронна комунікація за допомогою брокера повідомлень

Недоліком синхронної комунікації є те, що стан очікування між запитом і відповіддю значно знижує продуктивність додатку. Крім цього синхронна комунікація призводить до того, що одні мікросервіси починають залежати від інших мікросервісів, що є не ідеальним з точки зору мікросервісної архітектури.

Асинхронна комунікація за допомогою передачі повідомлень натомість має ряд переваг при роботі у розподілених системах[4]:

– передача повідомлень є стійкою до падінь мережі, бо системи передачі повідомлень зберігають повідомлення і відправляють їх тоді, коли доступність мережі відновлюється;
– система передачі повідомлень може гарантувати правильну обробку повідомлень, у випадку виникнення проблеми при обробці повідомлення пересилається заново;

– збільшується продуктивність додатку через те, що між запитом і відповіддю сервіс не блокується;
– відправник не повинен знати отримувача, що робить сервіси більш незалежними один від одного.

Тому було вирішено підтримувати генерацію конфігураційних файлів для розгортання брокерів повідомлень.

Майже кожний застосунок потребує зберігання даних. Для вирішення цієї задачі як правило використовуються бази даних. Згідно з принципами мікросервісної архітектури сервіси повинні бути незалежними один від одного. Для забезпечення повної незалежності кожний сервіс повинен зберігати свої дані окремо. На практиці спільні бази даних іноді використовуються, але вони мають ряд недоліків [4]:

– Зміни до представлення даних значно ускладнюються, через те що різні сервіси використовують одну базу даних і, відповідно, для зміни необхідна координація між командами усіх цих сервісів. Таким чином неможливо швидко змінити частину програми у випадку, якщо зміна включає у себе зміну схеми бази даних. В той же використання мікросервісної архітектури повинно було вирішити в тому числі саме цю проблему.
– Створюється можливість того, що через те що сервіси використовують одну і ту ж базу даних вони можуть втручатися у роботу один одного

Таким чином при розробці застосунку з використанням мікросервісної архітектури, кожен сервіс повинен зберігати свої власні дані. У випадку якщо сервіс потребує доступу до даних іншого сервісу, він може зробити це лише виконавши запит до API цього сервісу. Іншим варіантом є дуплікація даних і використання асинхронної комунікації, це дозволяє сервісам на залежати від доступності інших сервісів, але призводить до певних проблем, зокрема, з підтриманням цілісності даних.

Для окремого зберігання даних сервісу є різні варіанти, такі як:

– приватна таблиця у базі даних;
– приватна схема у базі даних;
– окрема база даних.

Тому було вирішено підтримувати генерацію конфігураційних файлів для розгортання баз даних.

1.3 Аналіз успішних IT-проектів

1.3.1 Аналіз відомих технічних рішень

Найпопулярнішими інструментами для розробки клієнтської частини додатку є бібліотека React та фреймворки Vue і Angular. Ці інструменти найкраще підходять для розробки односторінкових застосунків (single-page application). Суть такого клієнтського застосунку полягає у тому, що початковий HTML документ надсилається користувачу, а потім дані надсилаються користувачу невеликими частинами у відповідь на взаємодію з ним. Так як планується розробляти саме односторінковий застосунок, то необхідно вибрати один з цих трьох інструментів. Всі вони є досить схожими з точки зору функціональності [8], але є і ряд відмінностей(таблиця 1.3).

Таблиця 1.3 – Відмінності інструментів Angular[9], Vue[10], React[11]

Характеристика	Angular	Vue	React
MVC	Паттерн MVC використовується в якості стандартного способу структурування коду, що допомагає підтримувати складні застосунки		Не використовується

Продовження таблиці 1.3

Характеристика	Angular	Vue	React
Документація і підтримка	Наявна детальна документація, популярність фреймворку забезпечує якісну підтримку спільноти	Так як Vue має китайське коріння, значна частина спільноти є китайськомовною, що ускладнює пошук підтримки, має детальну документацію	Наявна детальна документація, популярність бібліотеки забезпечує якісну підтримку спільноти
Основна мова	Підтримує чистий Javascript, але для отримання всіх переваг від фреймворку необхідно використовувати Typescript	Підтримують як чистий Javascript, так і Typescipt	

Кінець таблиці 1.3

Характеристика	Angular	Vue	React
Структура застосунку	Має складну наперед визначену структуру застосунку, що складається з модулів, компонентів, сервісів і активно використовує ін’єкцію залежностей	Не накладає багатьох обмежень на структуру застосунку	Немає наперед визначеної структури
Важкість навчання	Висока	Низька	Низька

Для всіх трьох інструментів наявна детальна документація, а важкість навчання не має високого значення при розробці дипломного проекту. Таким чином головною різницею між інструментами є рівень структурованості, яку підтримує інструмент. Найкращим з цієї точки зору є Angular, тому було обрано його.

Для розробки серверної частини проекту є дуже багато варіантів можливих інструментів: ASP .NET, Spring Framework, Django, Ruby on Rails, ряд фреймворків для NodeJs (Express, Koa, Nest) тощо. Так як серверна частина проекту є невеликою і всі ці інструменти надають дуже схожу функціональність, то було вирішено обрати платформу NodeJs для розробки. Головною перевагою цього підходу порівняно з іншими інструментами є можливість розробляти клієнтську і серверну частину однією і тією ж мовою.

Найпопулярнішими фреймворками для серверної розробки на NodeJS є Express, Koa і Nest. При цьому Express і Koa є дуже схожими, так як Koa був розроблений розробниками Express. Nest[12] має ряд переваг над ними:

– наявність інтерфейсу командного рядка (CLI), який допомагає у створенні структурних елементів додатку і пришвидшує розробку;
– має наперед визначену структуру додатку, яка дуже схожа на структуру у фреймворку Angular, з активним використанням ін’єкції залежностей;
– сильна орієнтація на використання Typescript.

Тому було обрано Nest.

Для розробки інтерфейсу командного рядка підходить будь-яка мова загального використання: C++, C#, Java, Python, NodeJs. З усіх цих варіантів було обрано NodeJs, так як, по-перше, це дозволить писати весь проект однією мовою, і, по-друге, це дозволить за необхідності використати цей проект як залежність у інших частинах проекту, що дозволить використати оголошення певних класів на клієнті або сервері за необхідності.

1.3.2 Аналіз відомих програмних продуктів

Найближчим аналогом даного ПЗ є продукт JHipster. JHipster — це платформа для швидкого створення, розробки та розгортання сучасних веб-додатків, включаючи ті, що використовують мікросервісну архітектуру. JHipster підтримує велику кількість технологій (рисунок 1.6-1.9)

Рисунок 1.6 – Опції для клієнтської частини застосунку

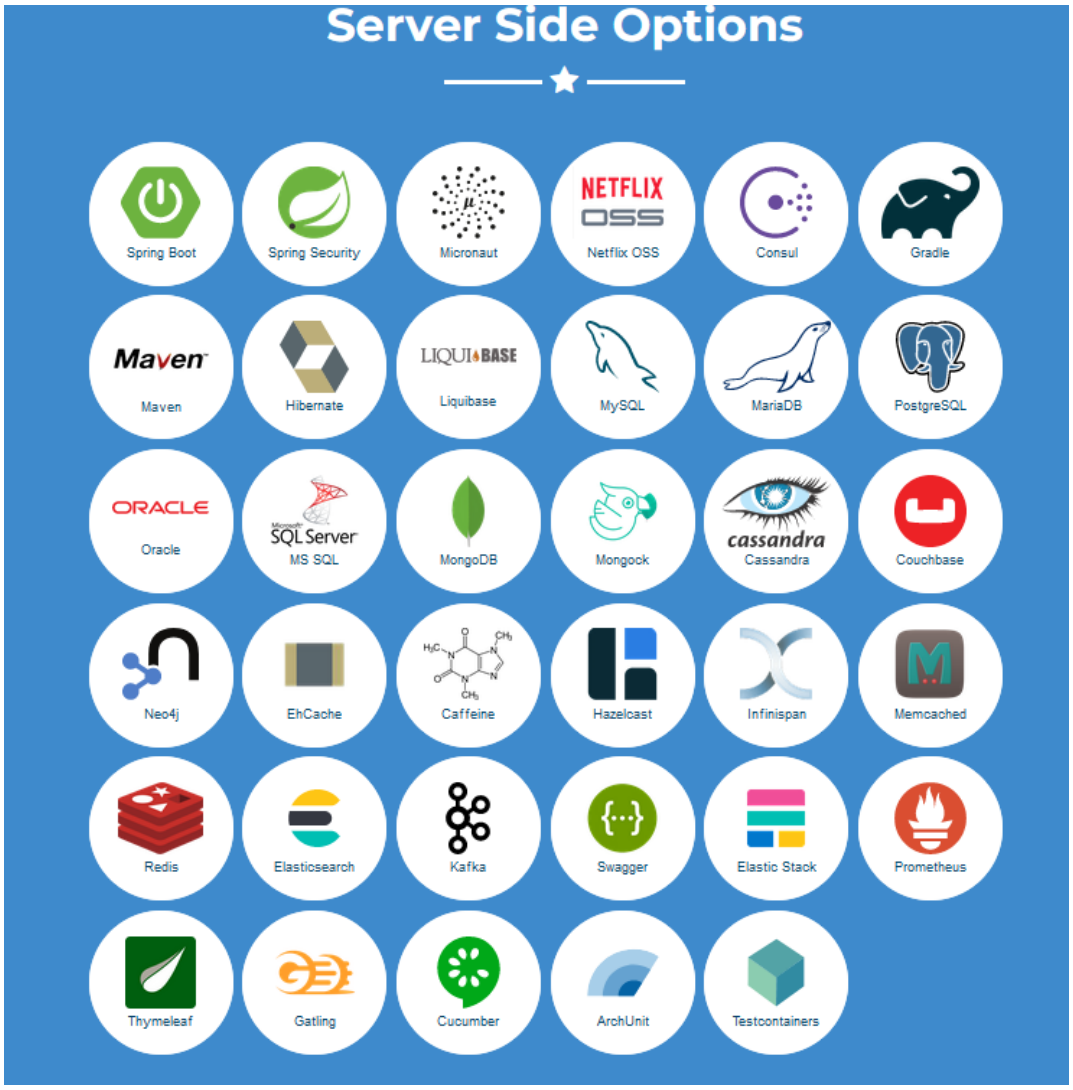


Рисунок 1.7 – Опції для серверної частини застосунку

Рисунок 1.8 – Опції для розгортання застосунку

Рисунок 1.9 – Опції для CI/CD

Для опису застосунку використовується JDL - DDL для опису конфігурації. Писати JDL можна у середовищах Eclipse, VS Code або JDL Studio (рисунок 1.10), який працює у web, та у реальному часі візуалізує схему застосунку. Після цього за допомогою CLI на основі конфігурації генеруються необхідні файли.

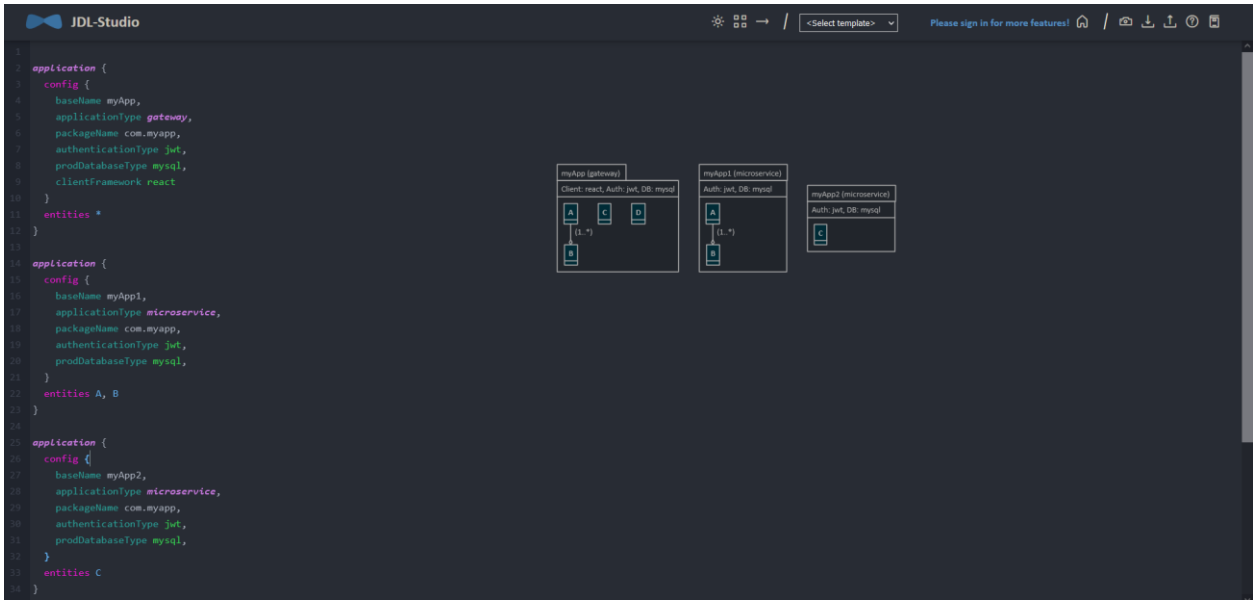


Рисунок 1.10 – JDL Studio

Порівняння функціональності JHipster і дипломного проекту наведене у таблиці 1.4.

Таблиця 1.4 Порівняння JHipster і дипломного проекту

Функціональність	JHipster	Дипломний проект
Генерація мікросервісів	Дозволяє генерувати мікросервіси, основою для генерації є опис даних	Дозволяє генерувати мікросервіси, основою для генерації є опис API
Генерація конфігураційних файлів для систем CI/CD	Дозволяє генерувати конфігураційні файли для систем CI/CD	Дозволяє генерувати конфігураційні файли для систем CI/CD
Генерація баз даних	Дозволяє генерувати бази даних, одна база даних завжди прив’язана до конкретного сервісу	Дозволяє генерувати бази даних, база даних може бути спільно використовуватись декількома сервісами, один сервіс може мати декілька баз даних
Генерація фронт-енду	Генерація фронт-енду підтримується	Генерація фронт-енду не підтримується

Продовження таблиці 1.4

Функціональність	JHipster	Дипломний проект
Візуальне редагування конфігурації	JHipster має окремий DDL для опису конфігурації, а також веб-редактор для нього, який у реальному часі візуалізує проект, проте JHipster не надає візуальних інструментів для редагування конфігурації	Наявний графічний інтерфейс, який дозволяє редагувати конфігурацію проекту без написання коду
Генерація конфігурації для розгортання брокера повідомлень	Генерація конфігурації для розгортання брокера повідомлень не підтримується	Генерація конфігурації для розгортання брокера повідомлень підтримується
Управління секретними значеннями (паролі, ключі тощо)	Управління секретними значеннями не підтримується	Під час процесу генерації макету застосунку програма отримує у користувача необхідні значення, створює відповідні секретні значення у системі CI/CD, та створює конфігураційні файли kubernetes, в яких описані ці секретні значення

1.4 Аналіз вимог до програмного забезпечення

1.4.1 Розроблення функціональних вимог

Проект повинен надавати користувачу можливість генерації шаблону застосунку з мікросервісною архітектурою, для цього користувача повинен за допомогою графічного інтерфейсу додати необхідні елементи, налаштувати їх і створити необхідні зв’язки між ними. Після цього користувач зможе завантажити конфігураційний файл і згенерувати шаблон застосунку за допомогою консольного застосунку. Діаграма варіантів використання наведена у додатках.

В таблицях 1.5 - 1.25 наведені варіанти використання програмного забезпечення.

Таблиця 1.5 - Варіант використання UC-1

Назва	Зареєструватись
Опис	Реєстрація нового користувача в системі
Учасники	Користувач
Постумови	Додавання нового користувача у системи, перехід на сторінку входу
Основний сценарій	Користувач переходить на сторінку реєстрації. В поля для реєстрації вводяться необхідні дані: пошта користувача, пароль, та підтвердження паролю для уникнення помилок. Після заповнення даних користувач натискає кнопку реєстрації. Після цього користувач перенаправляється на сторінку входу.
Розширення сценаріїв	Після заповнення даних користувач натискає кнопку реєстрації. Після цього у випадку якщо дані не пройшли всі необхідні перевірки, користувач отримує повідомлення про помилку з описом того, що йому треба виправити.

Таблиця 1.6 - Варіант використання UC-2

Назва	Авторизуватись
Опис	Авторизація користувача в системі
Учасники	Користувач
Постумови	Підтвердження особи користувача, перехід на сторінку управління проектами
Основний сценарій	Користувач переходить на сторінку входу. В поля для входу вводяться необхідні дані: пошта користувача та пароль. Після заповнення даних користувач натискає кнопку входу. Після цього користувач перенаправляється на сторінку управління проектами.
Розширення сценаріїв	Після заповнення даних користувач натискає кнопку входу. Після цього у випадку якщо пароль не відповідає пошті, користувач отримує повідомлення проте, що комбінація пошти і паролю є неправильною без додаткової інформації.

Таблиця 1.7 - Варіант використання UC-3

Назва	Створити проект
Опис	Створення нового проекту користувачем
Учасники	Користувач
Постумови	Створення нового проекту у системі з назвою за замовчуванням, перехід на сторінку конструювання проекту
Основний сценарій	Користувач переходить на сторінку управління проектами. Після цього користувач натискає кнопку створення нового проекту. Після цього користувач перенаправляється на сторінку конструювання проекту.
Розширення сценаріїв	-

Таблиця 1.8 - Варіант використання UC-4

Назва	Змінити назву проекту
Опис	Зміна назву проекту користувачем
Учасники	Користувач
Постумови	Зміна назви проекту у системі
Основний сценарій	Користувач переходить на сторінку конструювання проекту. Після цього користувач натискає на назву проекту і вводить нову назву.
Розширення сценаріїв	Якщо введена назва є некоректною, на панелі помилок з’являється повідомлення про помилку і не зникає допоки назва не буде виправлена.

Таблиця 1.9 - Варіант використання UC-5

Назва	Змінити конфігурації проекту
Опис	Зміна конфігурації проекту користувачем
Учасники	Користувач
Постумови	Зміна конфігурації проекту у системі
Основний сценарій	Користувач переходить на сторінку конструювання проекту. Після цього користувач натискає на кнопку редагування параметрів проекту, з’являється панель редагування конфігурації проекту. Користувач вводить необхідні дані, а саме обирає платформу для CI/CD, і вводить ім’я користувача dockerhub
Розширення сценаріїв	Якщо введені значеннями є некоректними, на панелі помилок з’являються повідомлення про всі помилки і не зникають допоки помилки не будуть виправлені.

Таблиця 1.10 - Варіант використання UC-6

Назва	Зберегти проект
Опис	Збереження проекту користувачем
Учасники	Користувач
Постумови	Збереження всіх змін проекту у системі
Основний сценарій	Користувач переходить на сторінку конструювання проекту. Після цього користувач натискає кнопку збереження проекту. Проект зберігається.
Розширення сценаріїв	У випадку наявності помилок у проекті, проект не зберігається, з’являється повідомлення про неможливість зберігання проекту за наявності помилок.

Таблиця 1.11 - Варіант використання UC-7

Назва	Вийти з проекту
Опис	Збереження проекту користувачем і вихід зі сторінки з проектом
Учасники	Користувач
Постумови	Збереження всіх змін проекту у системі, перехід на сторінку управління проектами
Основний сценарій	Користувач переходить на сторінку конструювання проекту. Після цього користувач натискає кнопку збереження і виходу. Проект зберігається, користувач перенаправляється на сторінку управління проектами.
Розширення сценаріїв	У випадку наявності помилок у проекті, проект не зберігається, з’являється повідомлення про неможливість зберігання проекту за наявності помилок.

Таблиця 1.12 - Варіант використання UC-8

Назва	Завантажити проект
Опис	Завантаження проекту користувачем
Учасники	Користувач
Постумови	Перехід на сторінку конструювання проекту з завантаженим проектом
Основний сценарій	Користувач переходить на сторінку управління проектами. Після цього користувач обирає бажаний проект зі списку і натискає кнопку завантаження. Користувач перенаправляється на сторінку конструювання проекту із завантаженим обраним проектом.
Розширення сценаріїв	-

Таблиця 1.13 - Варіант використання UC-9

Назва	Видалити проект
Опис	Видалення проекту користувачем
Учасники	Користувач
Постумови	Видалення проекту з системи, зникнення проекту зі списку проектів
Основний сценарій	Користувач переходить на сторінку управління проектами. Після цього користувач обирає бажаний проект зі списку і натискає кнопку видалення. Відкривається діалог, який просить підтвердження бажання видалити проект. Користувач натискає на кнопку підтвердження видалення. Проект видаляється зі списку проектів.
Розширення сценаріїв	Користувач натискає на кнопку відміни видалення. Проект не видаляється.

Таблиця 1.14 - Варіант використання UC-10

Назва	Створити елемент
Опис	Створення елементу користувачем
Учасники	Користувач
Постумови	Створення нового елементу у системі, поява елементу у робочій області
Основний сценарій	Користувач переходить на сторінку конструювання проекту. Після цього користувач обирає бажаний елемент на панелі інструментів зі списку і двічі натискає на нього. Елемент з’являється у робочій області.
Розширення сценаріїв	-

Таблиця 1.15 - Варіант використання UC-11

Назва	Створити мікросервіс
Опис	Створення мікросервісу користувачем
Учасники	Користувач
Постумови	Створення нового мікросервісу у системі, поява мікросервісу у робочій області
Основний сценарій	Користувач переходить на сторінку конструювання проекту. Після цього користувач обирає мікросервіс на панелі інструментів зі списку і двічі натискає на нього. Мікросервіс з’являється у робочій області.
Розширення сценаріїв	-

Таблиця 1.16 - Варіант використання UC-12

Назва	Створити базу даних
Опис	Створення бази даних користувачем
Учасники	Користувач
Постумови	Створення бази даних у системі, поява бази даних у робочій області
Основний сценарій	Користувач переходить на сторінку конструювання проекту. Після цього користувач обирає базу даних на панелі інструментів зі списку і двічі натискає на неї. База даних з’являється у робочій області.
Розширення сценаріїв	-

Таблиця 1.17 - Варіант використання UC-13

Назва	Створити API шлюз
Опис	Створення API шлюзу користувачем
Учасники	Користувач
Постумови	Створення API шлюзу у системі, поява API шлюзу у робочій області
Основний сценарій	Користувач переходить на сторінку конструювання проекту. Після цього користувач обирає API шлюз на панелі інструментів зі списку і двічі натискає на нього. API шлюз з’являється у робочій області.
Розширення сценаріїв	-

Таблиця 1.18 - Варіант використання UC-14

Назва	Створити брокер повідомлень
Опис	Створення брокера повідомлень користувачем
Учасники	Користувач
Постумови	Створення брокера повідомлень у системі, поява брокера повідомлень у робочій області
Основний сценарій	Користувач переходить на сторінку конструювання брокера повідомлень. Після цього користувач обирає брокер повідомлень на панелі інструментів зі списку і двічі натискає на нього. Брокер повідомлень з’являється у робочій області.
Розширення сценаріїв	-

Таблиця 1.19 - Варіант використання UC-15

Назва	Змінити налаштування елементу
Опис	Зміна налаштувань елементу користувачем
Учасники	Користувач
Постумови	Зміна налаштувань елементу у системі
Основний сценарій	Користувач переходить на сторінку конструювання проекту. Після цього користувач обирає бажаний елемент на робочій області і натискає на нього двічі. Відкривається панель редагування налаштувань елементу з відповідними полями в залежності від типу елемент (сервіс, база даних, API шлюз, брокер повідомлень). Користувач вводить бажані значення.
Розширення сценаріїв	Якщо введені значеннями є некоректними, на панелі помилок з’являються повідомлення про всі помилки і не зникають допоки помилки не будуть виправлені.

Таблиця 1.20 - Варіант використання UC-16

Назва	Змінити розташування елементу
Опис	Зміна розташування елементу на робочій області користувачем
Учасники	Користувач
Постумови	Зміна розташування елементу на робочій області
Основний сценарій	Користувач переходить на сторінку конструювання проекту. Після цього користувач обирає бажаний елемент на робочій області зі списку і перетягує його за верхню частину у потрібне місце.
Розширення сценаріїв	-

Таблиця 1.21 - Варіант використання UC-17

Назва	Видалити елемент
Опис	Видалення елементу користувачем
Учасники	Користувач
Постумови	Видалення елементу, зникнення елементу з робочої області, видалення усіх зв’язків пов’язаних з елементом, що видаляється
Основний сценарій	Користувач переходить на сторінку конструювання проекту. Після цього користувач обирає бажаний елемент на робочій області і натискає на нього двічі. Відкривається панель редагування налаштувань елементу. Користувач натискає кнопку видалення елементу. Відкривається діалог, який просить підтвердження бажання видалити елемент. Користувач натискає на кнопку підтвердження видалення. Елемент видаляється.
Розширення сценаріїв	Користувач натискає на кнопку відміни видалення. Елемент не видаляється.

Таблиця 1.22 - Варіант використання UC-18

Назва	Створити зв’язок між елементами
Опис	Створення зв’язку між елементами користувачем
Учасники	Користувач
Постумови	Створення зв’язку між елементами, відображення лінії між елементами на робочій області.
Основний сценарій	Користувач переходить на сторінку конструювання проекту. Після цього користувач обирає бажаний елемент на робочій області натискає на нього і перетягує курсор до елементу, з яким він бажає створити зв’язок і відпускає мишу.
Розширення сценаріїв	У випадку якщо елементи не є сумісними зв’язок не створюється, і на панелі помилок з’являється повідомлення про помилку.

Таблиця 1.23 - Варіант використання UC-19

Назва	Видалення усіх зв’язків елементу
Опис	Видалення усіх зв’язків елементу користувачем
Учасники	Користувач
Постумови	Видалення усіх зв’язків елементу, зникнення ліній, що відображали ці зв’язки.
Основний сценарій	Користувач переходить на сторінку конструювання проекту. Після цього користувач обирає бажаний елемент на робочій області і натискає на нього двічі. Відкривається панель редагування налаштувань елементу. Користувач натискає кнопку видалення усіх зв’язків елементу. Відкривається діалог, який просить підтвердження бажання видалити усі зв’язки елементу. Користувач натискає на кнопку підтвердження видалення. Усі зв’язки елементу видаляються.
Розширення сценаріїв	Користувач натискає на кнопку відміни видалення. Зв’язки елементу не видаляються.

Таблиця 1.24 - Варіант використання UC-20

Назва	Отримати конфігураційний файл з сайту
Опис	Отримання конфігураційного файлу користувачем
Учасники	Користувач
Постумови	Завантаження конфігураційного файлу користувачем
Основний сценарій	Користувач переходить на сторінку конструювання проекту. Після цього користувач натискає кнопку завантаження конфігураційного файлу. Відбувається завантаження конфігураційного файлу.
Розширення сценаріїв	Якщо у проекті наявні помилки, то завантаження не відбувається, а користувач отримує повідомлення про те, що завантаження конфігураційного файлу неможливе за наявності помилок у проекті.

Таблиця 1.25 - Варіант використання UC-21

Назва	Отримати згенерований макет застосунку
Опис	Отримання згенерованого макету застосунку користувачем
Учасники	Користувач
Постумови	Отримання згенерованого макету застосунку користувачем
Основний сценарій	Користувач запускає CLI. У команді він вказує розташування конфігураційного файлу та місце куди зберегти проект. Відбувається генерація проекту. Після цього користувача просять ввести секретні значення (паролі від баз даних, ключі для взяємодії з системою CI/CD тощо), він вводить, генерація завершується.
Розширення сценаріїв	Якщо у проекті наявні помилки, то генерація не відбувається, а користувач отримує повідомлення про помилку.

Функціональні вимоги до програмного забезпечення наведені у таблицях 1.26-1.51. Матрицю трасування вимог можна побачити на рисунку 1.12.

Таблиця 1.26 – Функціональна вимога FR-1

Назва	Реєстрація користувача
Опис	Система повинна надавати можливість реєстрації користувача вимагаючи від нього такі дані: електронна пошта, пароль, підтвердження паролю

Таблиця 1.27 – Функціональна вимога FR-2

Назва	Авторизація користувача
Опис	Система повинна надавати можливість авторизації користувача у системі вимагаючи від нього такі дані: електронна пошта, пароль

Таблиця 1.28 – Функціональна вимога FR-3

Назва	Створення проекту
Опис	Система повинна надавати можливість користувачу створити новий проект

Таблиця 1.29 – Функціональна вимога FR-4

Назва	Зміна назви проекту
Опис	Система повинна надавати можливість користувачу змінити назву проекту

Таблиця 1.30 – Функціональна вимога FR-5

Назва	Зміна конфігурації проекту
Опис	Система повинна надавати можливість користувачу змінити конфігурацію проекту, а саме: обрати CI/CD систему, вказати користувацьке ім’я у DockerHub

Таблиця 1.31 – Функціональна вимога FR-6

Назва	Збереження проекту
Опис	Система повинна надавати можливість користувачу зберегти проект, а саме зберегти конфігурацію проекту та конфігурацію усіх елементів (сервісів, баз даних, брокерів повідомлень, і API шлюзів)

Таблиця 1.32 – Функціональна вимога FR-7

Назва	Закриття проекту
Опис	Система повинна надавати можливість користувачу закрити проект

Таблиця 1.33 – Функціональна вимога FR-8

Назва	Завантаження проекту
Опис	Система повинна надавати можливість користувачу завантажити раніше збережений проект

Таблиця 1.34 – Функціональна вимога FR-9

Назва	Видалення проекту
Опис	Система повинна надавати можливість користувачу видалити раніше створений проект

Таблиця 1.35 – Функціональна вимога FR-10

Назва	Додавання мікросервісу
Опис	Система повинна надавати можливість користувачу додати у проект мікросервіс

Таблиця 1.36 – Функціональна вимога FR-11

Назва	Додавання бази даних
Опис	Система повинна надавати можливість користувачу додати у проект базу даних

Таблиця 1.37 – Функціональна вимога FR-12

Назва	Додавання брокеру повідомлень
Опис	Система повинна надавати можливість користувачу додати у проект брокер повідомлень

Таблиця 1.38 – Функціональна вимога FR-13

Назва	Додавання API шлюзу
Опис	Система повинна надавати можливість користувачу додати у проект API шлюз

Таблиця 1.39 – Функціональна вимога FR-14

Назва	Редагування мікросервісу
Опис	Система повинна надавати можливість користувачу змінити параметри доданого у проект мікросервісу, а саме змінити назву, мову, порт, кількість реплік та конфігурацію API мікросервісу а також обрати чи генерувати документацію API для мікросервісу

Таблиця 1.40 – Функціональна вимога FR-15

Назва	Редагування бази даних
Опис	Система повинна надавати можливість користувачу змінити параметри доданої у проект бази даних, а саме змінити назву, тип і кількість місця виділеного для бази даних

Таблиця 1.41 – Функціональна вимога FR-16

Назва	Редагування брокера повідомлень
Опис	Система повинна надавати можливість користувачу змінити параметри доданого у проект брокера повідомлень, а саме обрати тип і кількість реплік брокера повідомлень

Таблиця 1.42 – Функціональна вимога FR-17

Назва	Редагування API шлюза
Опис	Система повинна надавати можливість користувачу змінити параметри доданого у проект API шлюза, а саме обрати ім’я хоста

Таблиця 1.43 – Функціональна вимога FR-18

Назва	Зміна розташування елементу проекту
Опис	Система повинна надавати можливість користувачу змінити розташування елементу проекту на робочій області

Таблиця 1.44 – Функціональна вимога FR-19

Назва	Видалення елементу проекту
Опис	Система повинна надавати можливість користувачу видалити створений раніше елемент

Таблиця 1.45 – Функціональна вимога FR-20

Назва	Створення зв’язку між сервісами
Опис	Система повинна надавати можливість користувачу створити зв’язок між двома сервісами

Таблиця 1.46 – Функціональна вимога FR-21

Назва	Створення зв’язку між сервісом і базою даних
Опис	Система повинна надавати можливість користувачу створити зв’язок між сервісом і базою даних

Таблиця 1.47 – Функціональна вимога FR-22

Назва	Створення зв’язку між сервісом і брокером повідомлень
Опис	Система повинна надавати можливість користувачу створити зв’язок між сервісом і брокером повідомлень

Таблиця 1.48 – Функціональна вимога FR-23

Назва	Створення зв’язку між сервісом і API шлюзом
Опис	Система повинна надавати можливість користувачу створити зв’язок між сервісом і API шлюзом

Таблиця 1.49 – Функціональна вимога FR-24

Назва	Видалення усіх зв’язків елементу
Опис	Система повинна надавати можливість користувачу видалити усі зв’язки елемента

Таблиця 1.50 – Функціональна вимога FR-25

Назва	Завантаження конфігураційного файлу
Опис	Система повинна надавати можливість користувачу завантажити конфігураційний файл з описом проекту

Таблиця 1.51 – Функціональна вимога FR-26

Назва	Генерація макету застосунку
Опис	Система повинна надавати можливість користувачу згенерувати макет застосунку з мікросервісною архітектурою за наявним конфігураційним файлом з описом проекту

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17	FR-18	FR-19	FR-20	FR-21	FR-22	FR-23	FR-24	FR-25	FR-26
UC-1	+																									
UC-2		+																								
UC-3			+																							
UC-4				+																						
UC-5					+																					
UC-6						+																				
UC-7							+																			
UC-8							+	+																		
UC-9									+																	
UC-10										+	+	+	+													
UC-11										+																
UC-12											+															
UC-13												+														
UC-14													+													
UC-15														+	+	+	+									
UC-16																		+								
UC-17																			+							
UC-18																				+	+	+	+			
UC-19																								+		
UC-20																									+	
UC-21																										+

Рисунок 2.12 - Матриця трасування вимог

1.4.2 Розроблення нефункціональних вимог

Програмне забезпечення повинно:

– мати зрозумілий інтерфейс;

– бути легким у використанні;

– показувати нормальну продуктивність роботи на сучасних браузерах;

– бути сумісним з сучасними браузерами;

– бути захищеним від кібер-атак;

1.4.3 Постановка комплексу завдань модулю

Програмне забезпечення повинно надавати користувачу можливість генерування макетів застосунків з мікросервісною архітектурою. Для цього програма надає можливість користувачу за допомогою графічного інтерфейсу створити і налаштувати елементи застосунку з мікросервісною архітектури, а саме мікросервіси, бази даних, брокери повідомлень, API шлюзи. Після цього програма генерує конфігураційний файл на основі якого можна згенерувати макет застосунку.

Метою розробки є:

– спрощення і пришвидшення розробки застосунку з мікросервісною архітектурою;

– зменшення витрат необхідних на початкових етапах розробки;

– зменшення ймовірності помилок при розробці застосунку з мікросервісною архітектурою.

Розробка призначена для:

– веб-розробників і DevOps інженерів, які розробляють веб-застосунки з використанням мікросервісної архітектури;

– студентів, які вивчають інструменти для розробки веб-застосунків з мікросервісною архітектурою.

Мета досягається за рахунок розв’язання наступних задач:

– реалізації інструментів для реєстрації користувача;

– реалізації інструментів для авторизації користувача;

– реалізації інструментів для створення, збереження, завантаження і видалення проектів користувача;

– створення графічного інтерфейсу для створення і зміни описів мікросервісів, баз даних, брокерів повідомлень і API шлюзів, з яких складається застосунок;

– створення інструментів для експорту проекту у вигляді конфігураційного файлу для консольного застосунку, який виконує генерацію макету веб-додатку;

– створення інструментів для генерації конфігураційних файлів kubernetes для розгортання мікросервісів, баз даних, брокерів повідомлень і API шлюзів;

– створення інструментів для генерації команд для створення хмарної інфраструктури;

– створення інструментів для генерації макету мікросервісу на базі опису API;

– створення інструментів для генерації документації з описом API на базі OpenAPI.

Висновки до розділу

В першому розділі дипломної роботи було досліджено і проаналізовано предметну область програмного забезпечення. Аналіз показав, що для створення застосунку з мікросервісною архітектурою корисно мати засіб, який би згенерував:

– конфігураційні файли Docker;

– конфігураційні файли kubernetes для розгортання сервісів;

– конфігураційні файли kubernetes для розгортання баз даних;

– конфігураційні файли kubernetes для розгортання брокерів повідомлень.

Були проаналізовані аналоги програмного забезпечення. Аналіз аналогів показав, що розробка має ряд переваг:

– наявність візуального редагування параметрів елементів застосунку;

– можливість генерації конфігураційних файлів для розгортання брокерів повідомлень;

– можливість генерації файлів для управління секретними значеннями проекту.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Для опису бізнес-процесів реєстрації користувача і генерації користувачем макету застосунку з мікросервісною архітектурою використовуються BPMN діаграми (рисунок 2.1 і 2.2). Опис послідовності дій користувача:

– якщо користувач не зареєстрований, він реєструється;

– користувач авторизується у веб-додатку;

– користувач створює новий проект;

– користувач створює необхідні елементи;

– користувач налаштовує елементи;

– користувач завантажує конфігураційний файл з описом проекту;

– користувач за допомогою консольного застосунку генерує макета застосунку з мікросервісною архітектурою, вказавши шлях до конфігураційного файлу і шлях збереження проекту.

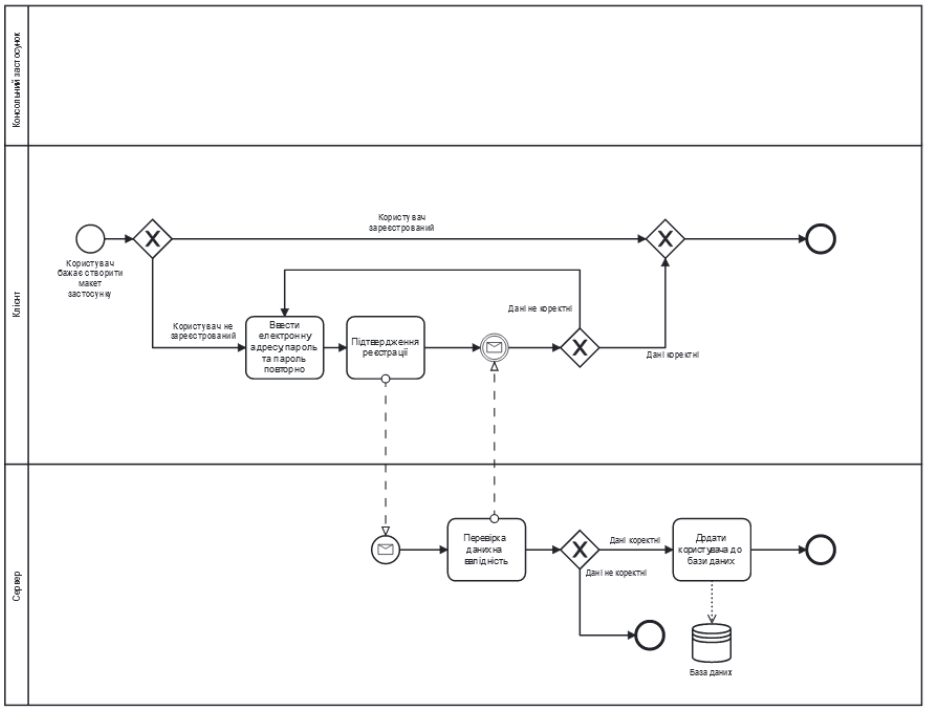


Рисунок 2.1 – BPMN діаграма реєстрації користувача

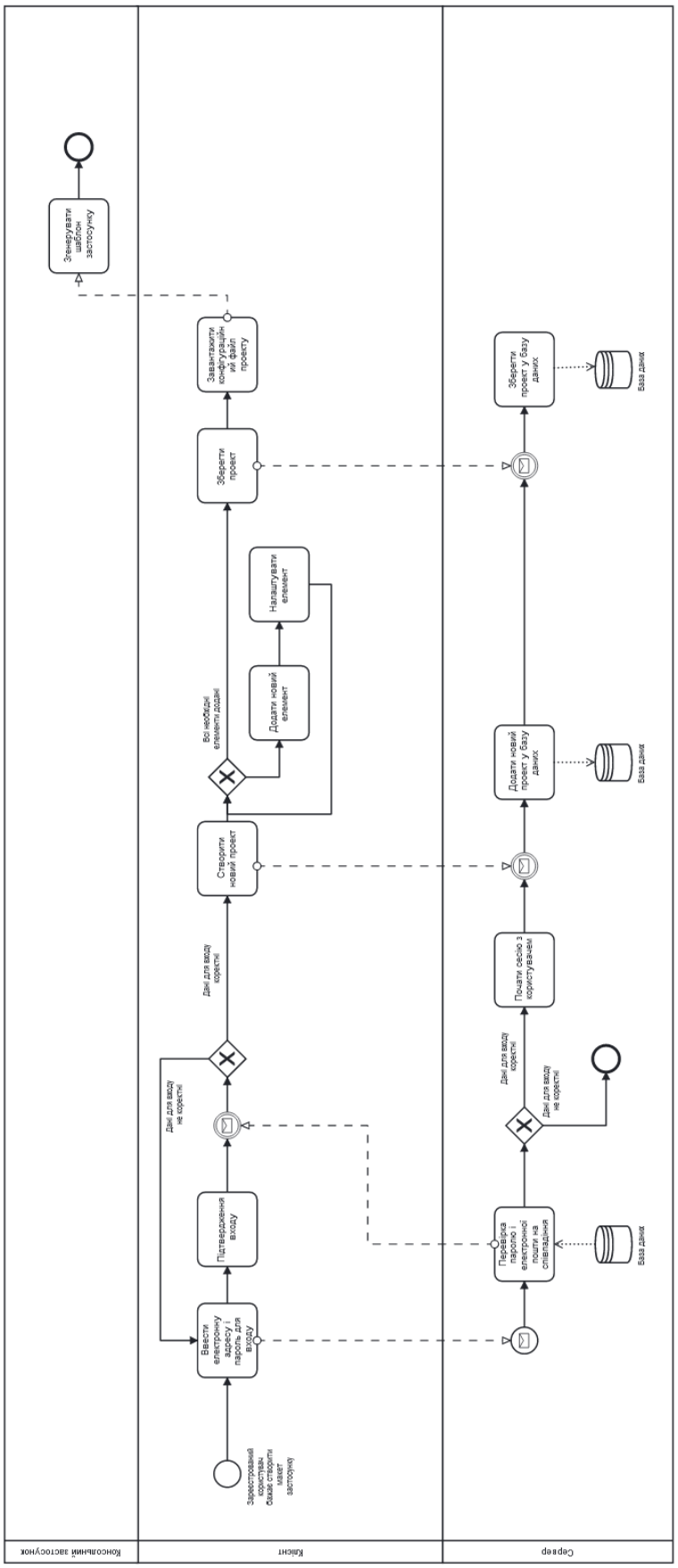


Рисунок 2.2 – BPMN діаграма конструювання користувачем макету застосунку

2.2 Архітектура і конструювання програмного забезпечення

У склад програмного забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою входять веб-застосунок і консольний застосунок (рисунок 2.3).

Веб-застосунок надає користувачу графічний інтерфейс для конструювання опису бажаного застосунку і можливість отримати конфігураційний файл з цим описом. При цьому веб-застосунок має клієнтську і серверну частини.

Консольний додаток приймає на вхід конфігураційний файл з описом бажаного застосунку і генерує його макет. Макет включає конфігураційні файли для розгортання мікросервісів, баз даних, брокерів повідомлень, шлюзів API, а також файли з частиною коду самих мікросервісів.

Рисунок 2.3 – Схема взаємодія частин ПЗ

2.2.1 Архітектура і конструювання консольного застосунку

Консольний застосунок отримує на вхід конфігураційний файл, що містить опис проекту у форматі JSON, структура цього файлу зображена на рисунку 2.4.

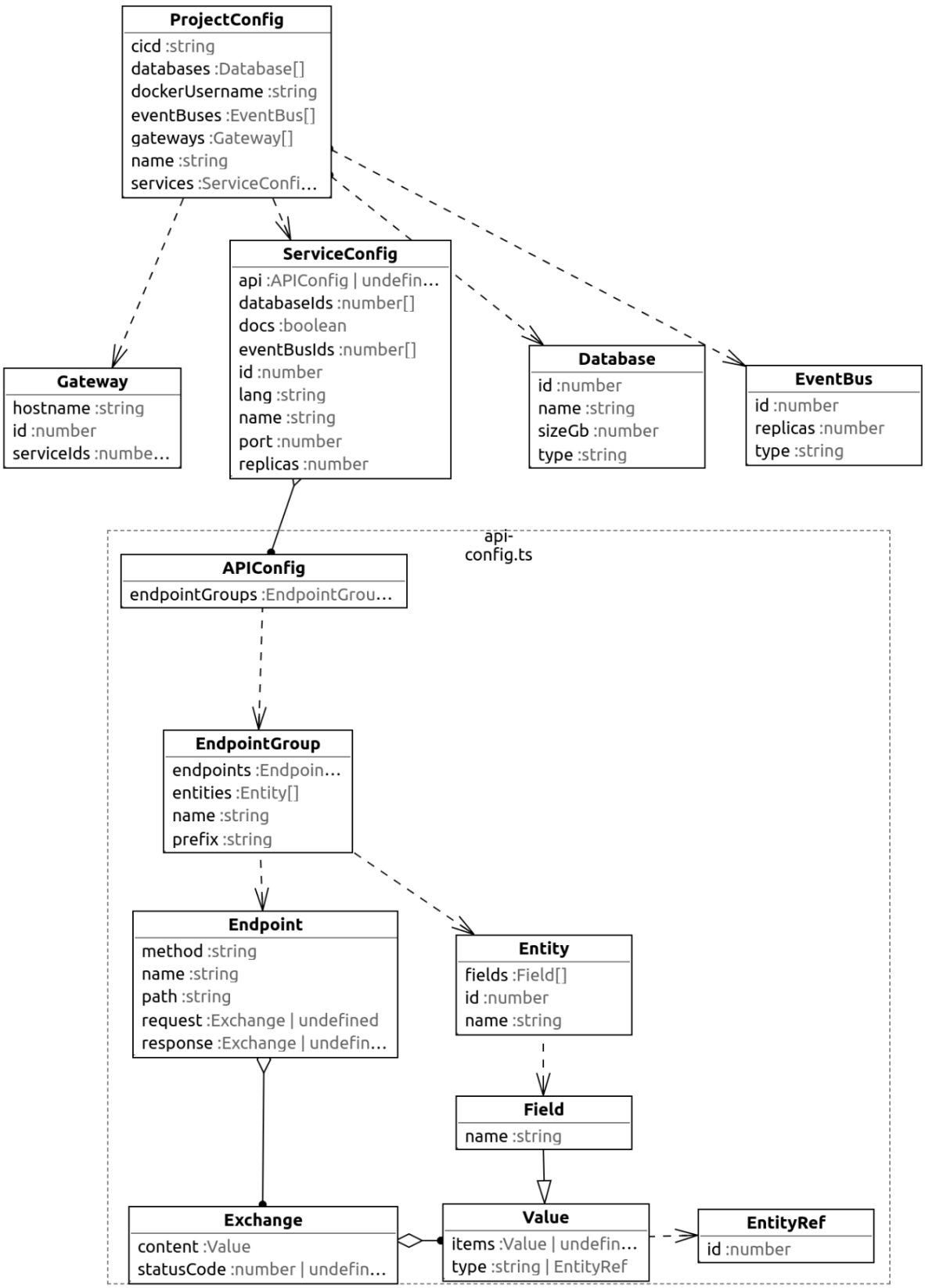


Рисунок 2.4 – Структура конфігураційного файлу

Бібліотеки, які використовуються у консольному додатку вказані у таблиці 2.1.

Таблиця 2.1 - Залежності

Залежність	Опис
typescript	залежність що дозволяє писати мовою typescript, що надає програмі статичну типізацію, потім програма транспілюється у Javascript
eslint	лінтер, дозволяє перевіряти код на відповідність визначеному стилю
prompt	бібліотека, що полегшує роботу з вводом користувача у консоль
ts-command-line-args	бібліотека, що полегшує роботу з аргументами командного рядка
reflect-metadata	розширює можливості Typescript з рефлексії

Для генерації програми використовуються класи, що імплементують інтерфейс FileGenerator<T>, вони отримують аргумент типу T і генерують відповідні файли. Для деяких елементів клас треба обрати з декількох варіантів в залежності від типу елементу, так для бази даних треба буде обрати між класами PostgresDatabaseGenerator і MongoDatabaseGenerator. Для цієї задачі існує клас GeneratorFactory<T> який у конструктор приймає асоціативний масив який встановлює відповідність між типами елементу і класами.

Кожен з генераторів файлів має у собі ряд класів, що імплементують інтерфейс FileTemplate, що повертає один файл. При генерації файлів часто можна обійтись лише рядковими шаблонами Javascript, але у ряді випадків

таких як генерація контролеру для групи ендпоінтів потрібна більш складна логіка.

У функції generateProject(config: ProjectConfig) генеруються всі необхідні файли та передаються у клас FileWriter, який приймає масив файлів та записує їх у файлову систему. Під файлами при цьому маються на увазі об’єкти, що належать класу File (таблиця 2.2)

Таблиця 2.2 - Опис класу File

Поле	Тип	Опис
name	string	назва файлу
path	string	шлях у який треба записати файл
data	string	дані які треба записати у файл

Також класи, що імплементують інтерфейс FileGenerator мають доступ до класу SecretsCreator, це дає змогу у цьому класі створити за бажанням певну кількість секретів (паролі, ключі тощо). Після цього SecretsCreator просить користувача ввести їх значення і створює .sh файл, який надає секретам відповідні значення, а також створює команди для того, щоб створити ці секрети у системі CI/CD.

Для того, щоб забезпечити легке додавання нових елементів було створено ряд декораторів (таблиця 2.3), які надають необхідні дані(таблиця 2.4) про поля налаштувань елементів і проекту для того, щоб на клієнтській частині програми інтерфейс заповнення цих полів міг бути згенерований динамічно в залежності від декларацій саме на консольному додатку. Це допоможе уникнути проблем з сумісністю які мали б високий ризик виникнення у випадку, якщо б додавати нові елементи на клієнт, сервер і консольний додаток треба було б окремо. Приклад використання цих декораторів можна побачити на рисунку 2.5 і на рисунку 2.6.

Таблиця 2.3 - Декоратори

Декоратор	Опис
StringField	визначає поле як рядок
NumberFeild	визначає поле як номер
BooleanField	визначає поле як логічне значення
CustomField	визначає поле як таке, що має в якості типу певний клас, наприклад, APIConfig
EnumField	визначає поле як таке, що може приймати певну кількість наперед визначених значень
IdField	визначає поле як ідентифікаційний номер
ReferenceField	визначає поле як набір посилань на ідентифікаційні номери інших елементів
Entity	визначає поле як таке, що містить набір елементів з їх полями

Таблиця 2.4 – Опис класу FieldData

Поле	Тип	Опціональне	Опис
name	string	-	назва поля
type	string	-	тип поля
isId	boolean	-	флаг, що показує чи є поле ідентифікаційним номером

Продовження таблиці 2.4

Поле	Тип	Опціональне	Опис
possibleValues	string[]	+	показує набір можливих значень для поля
references	string[]	+	показує типи елементів на які може посилатись це поле

```
export class ProjectConfig {
  @StringField
  name: string;

  @EnumField(availableCICDSetups)
  cicd: string;

  @StringField
  dockerUsername: string;

  @Entity('gateway', Gateway)
  gateways: Gateway[];

  @Entity('eventBus', EventBus)
  eventBuses: EventBus[];

  @Entity('service', ServiceConfig)
  services: ServiceConfig[];

  @Entity('database', Database)
  databases: Database[];
}
```

Рисунок 2.5 – Використання декораторів для опису полів конфігурації проекту

```
export class ServiceConfig {
  @IdField
  id: number;

  @StringField
  name: string;

  @EnumField(availableServiceSetups)
  lang: string;

  @NumberField
  port: number;

  @BooleanField
  docs: boolean;

  @NumberField
  replicas: number;

  @ReferenceField(['database'])
  databaseIds: number[];

  @ReferenceField(['eventBus'])
  eventBusIds: number[];

  @CustomTypeField('APIConfig')
  api?: APIConfig;
}
```

Рисунок 2.6 – Використання декораторів для опису полів конфігурації сервісу

2.2.2 Архітектура і конструювання клієнтської частини проекту

Для завантаження бібліотек та інших залежностей використовується менеджер пакетів npm (node package manager). Імпортовані бібліотеки вказані у таблиці 2.5.

Таблиця 2.5 – Залежності клієнтської частини проекту

Залежність	Опис
angular	фреймворк для створення веб-застосунків
rxjs	бібліотека для створення асинхронних систем на основі подій, з широким використанням паттернів observer і observable
typescript	залежність що дозволяє писати мовою typescript, що надає програмі статичну типізацію, потім програма транспілюється у Javascript
eslint	лінтер, дозволяє перевіряти код на відповідність визначеному стилю

У фреймворці Angular основними будівельними блоками є модулі, компоненти і сервіси. Наведемо схему з їх зображенням (рисунок 2.7), а також опис роботи основних компонентів (таблиця 2.6.) і сервісів (таблиця 2.7.)

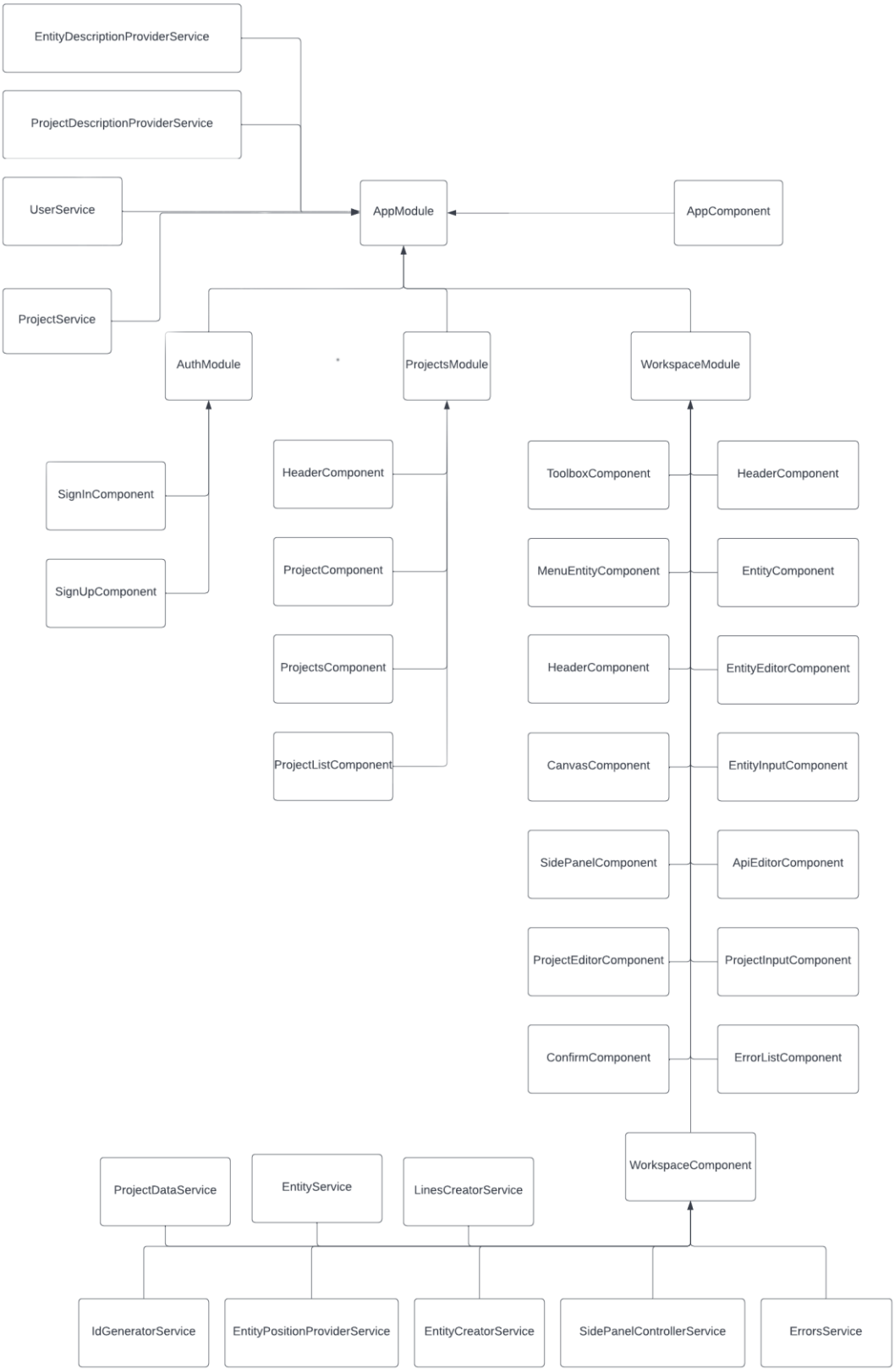


Рисунок 2.7 – Структура клієнту

Таблиця 2.6 – Компоненти клієнтської частини програми

Компонент	Опис
AppComponent	уміщає у себе компонент, який відповідає поточній адресі
SignUpComponent	містить форму реєстрації
SignInComponent	містить форму авторизації
HeaderComponent (ProjectsModule)	містить назву програми та кнопку для виходу з аккаунту
ProjectsComponent	містить усю сторінку для управління проектами, а саме заголовок, список проектів, обраний проект і кнопки для взаємодії з ним, відображає кнопку для створення нового проекту
ProjectListComponent	містить список проектів, дозволяє обирати проект для подальшої взаємодії
ProjectComponent	відображає обраний проект, а саме його назву, дату створення та останнього збереження, відображає кнопки для завантаження та видалення проекту
WorkspaceComponent	відображає усю сторінку конструювання проекту
HeaderComponent	відображає заголовок, який можна редагувати і кнопки для взаємодії з проектом, а саме кнопки для збереження проекту, виходу з проекту, збереження і виходу з проекту, відкриття панелі редагування конфігурації проекту, експортування конфігураційного файлу

Продовження таблиці 2.6

Компонент	Опис
ToolboxComponent	відображає набір елементів, які за подвійним натисканням створюються на робочій області
MenuEntityComponent	відображає елемент який за подвійним натисканням створюється на робочій області
EntityComponent	відображає елемент на робочій області, який можна по ній переміщати, також можна створювати зв’язки між елементами і обирати елемент для редагування
EntityEditorComponent	дозволяє редагувати налаштування обраного елементу
EntityInputComponent	відображає поле редагування для конкретного налаштування обраного елементу
ProjectEditorComponent	дозволяє редагувати налаштування проекту
ProjectInputComponent	відображає поле редагування для конкретного налаштування проекту
CanvasComponent	відображає робочу область де знаходяться елементи і зв’язки між ними
ApiEditorComponent	відображає форму для редагування конфігурації API обраного сервісу
SidePanelComponent	відображає бокову панель, яка в залежності від стану може або не відображатись взагалі, або містити інші компоненти

Кінець таблиці 2.6

Компонент	Опис
ConfirmComponent	відображає набір діалогове вікно, що пропонує підтвердити дію або відмінити її
ErrorListComponent	відображає список помилок у проекті

Таблиця 2.7 – Сервіси клієнтської частини додатку

Сервіс	Опис
ProjectDescriptionProviderService	надає опис полів конфігурації проекту, для того, щоб можна було динамічно сконструювати форму для їх зміни, використовує консольний застосунок як залежність для того, щоб отримувати цей опис
EntityDescriptionProviderService	надає опис полів конфігурації елементів (сервісів, баз даних, брокерів повідомлень, API шлюзів), для того, щоб можна було динамічно сконструювати форму для їх зміни, використовує консольний застосунок як залежність для того, щоб отримувати цей опис, це дозволяє уникнути проблем з сумісністю між консольним застосунком і клієнтом
UserService	надсилає запити на сервер, що пов’язані з реєстрацією і авторизацією

Продовження таблиці 2.7

Сервіс	Опис
ProjectDataService	керує даними проекту і відслідковує їх зміну користувачем, також займається формуванням даних для конфігураційного файлу
EntityService	керує даними елементів і відслідковує їх зміну користувачем, також займається підготовкою даних до зберігання на сервер
LinesCreatorService	дозволяє створювати і видаляти лінії на робочій області
IdGeneratorService	займається генерацією ідентифікаційних номерів для елементів
EntityPositionProviderService	відслідковує чи знаходиться курсор на якомусь елементі і на якому саме
EntityCreatorService	керує створенням нових елементів
SidePanelControllerService	керує станом бокової панелі
ProjectService	надсилає запити на сервер, що пов’язані з управлінням проектами, такі як створення нового проекту, видалення і зберігання проекту, отримання списку проектів
ErrorsSerivce	дозволяє іншим компонентам або сервісам додавати повідомлення про помилки

2.2.3 Архітектура і конструювання серверної частини проекту

Наведемо залежності проекту у таблиці 2.8. Основною з залежностей є фреймворк NestJs. У фреймворці NestJs основними будівельними блоками є модулі, контролери і провайдери. Наведемо схему з їх зображенням (рисунок 2.8), а також опис роботи основних контролери (таблиця 2.9) і провайдерів (таблиця 2.10)

Таблиця 2.1. – Залежності серверної частини проекту

Залежність	Опис
nestjs	фреймворк для створення серверних застосунків
nestjs/passport	бібліотека passport для вирішення задачі авторизації, адаптована для фреймворку nestjs
nestjs/throttler	додає можливість встановлювати ліміт на кількість запитів, що приходять, дозволяє зменшити ризики для безпеки пов’язані зі спробами підібрати паролі до аккаунтів користувачів
argon2	реалізує хешування за допомогою argon2, є одним з найкращих алгоритмів хешування
class-validator	дозволяє додавати валідацію даних що приходять на сервер за допомогою декораторів
express-session	дозволяє створювати і керувати користувацькими сесіями
knex	є бібліотекою, що дозволяє будувати запити до бази даних
pg	дозволяє взаємодіяти з базою даних Postgres

Продовження таблиці 2.8

Залежність	Опис
rxjs	бібліотека для створення асинхронних систем на основі подій, з широким використанням паттернів observer і observable
typescript	залежність що дозволяє писати мовою typescript, що надає програмі статичну типізацію, потім програма транспілюється у Javascript
eslint	лінтер, дозволяє перевіряти код на відповідність визначеному стилю
jest	тестовий фреймворк для Javascript
supertest	бібліотека для тестування HTTP сервісів

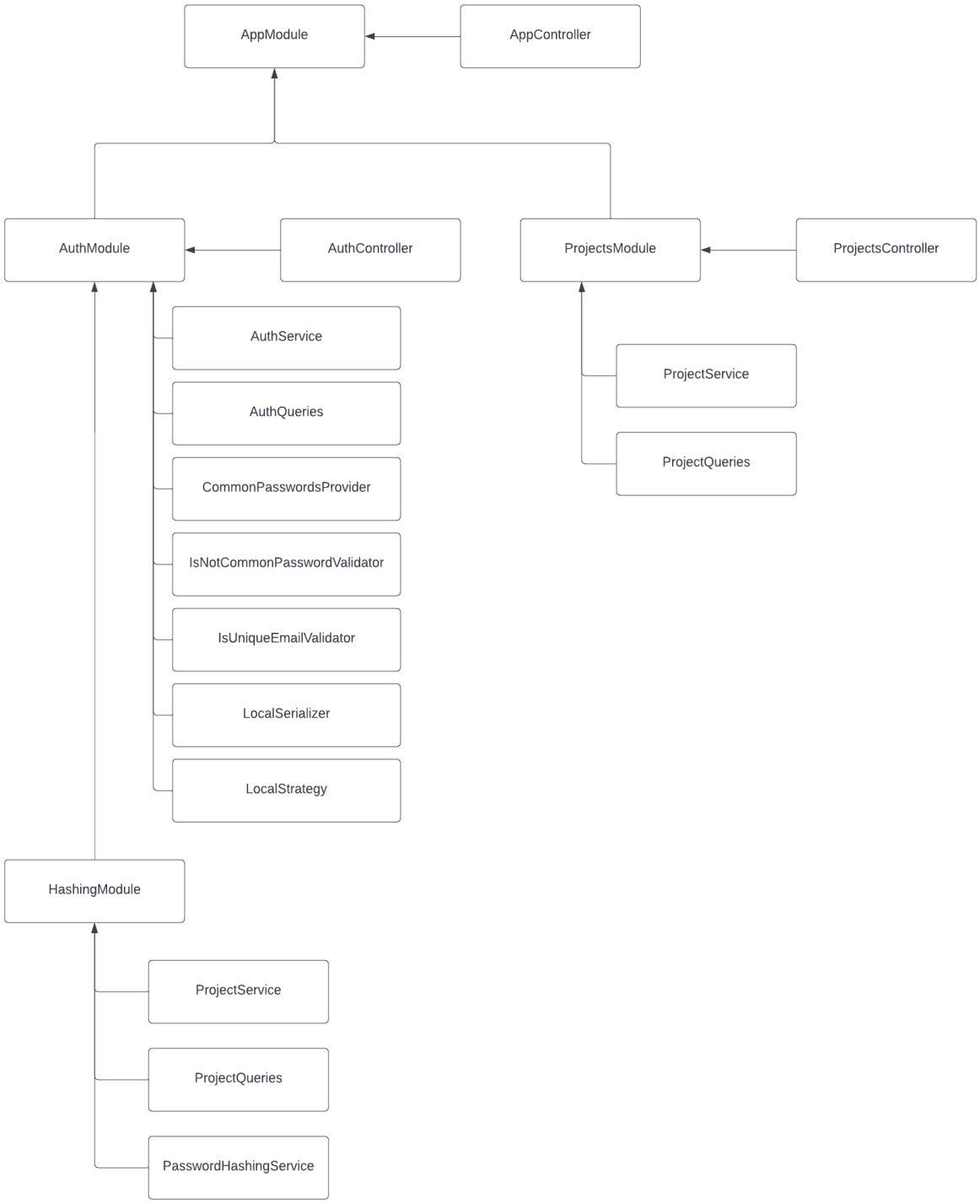


Рисунок 2.8 – Структура серверу

Таблиця 2.9 – Провайдери серверу

Сервіс	Опис
AuthService	оброблює логіку пов’язану з реєстрацією і авторизацією
AuthQueries	виконує запити до бази даних, які пов’язані з реєстрацією і авторизацією
CommonPasswordsProvider	дозволяє отримати певну кількість найбільш популярних паролей (кількість встановлюється у конфігурації)
IsNotCommonPasswordValidator	декоратор валідатор, що дозволяє перевіряти поле паролю на те що він не є одним з певної кількості найбільш популярних паролей
IsUniqueEmailValidator	декоратор валідатор, що дозволяє перевіряти чи не існує на момент реєстрації аккаунт з такою самою електронною поштою
LocalSerializer	серіалізує і десеріалізує користувача для авторизації
HasherFactory	дозволяє отримати необхідний клас, який буде виконувати хешування
HashingQueries	дозволяє виконувати запити у базу даних пов’язані з хешуванням

Продовження таблиці 2.9

Сервіс	Опис
PasswordHashingService	хешує паролі, перевіряє хеш, також у випадку зміни схеми хешування тимчасово хешує уже існуючі хеші паролей, а у випадку якщо користувач успішно заходить оновлює захешований пароль і позбувається вкладених хешів
ProjectService	оброблює логіку пов’язану зі створенням, збереженням і видаленням проектів
ProjectQueries	виконує запити до бази даних, які пов’язані зі створенням, збереженням і видаленням проектів

Таблиця 2.10 – Контролери серверу

Контролер: AppController		
GET	/	підтверджує роботу сервісу
Контролер: AuthController		
POST	/auth/register	виконує реєстрацію
POST	/auth/login	виконує авторизацію
GET	/auth/me	повертає дані про користувача
POST	/auth/logout	дозволяє закінчити сесію

Продовження таблиці 2.10

Контролер: ProjectController		
GET	/projects	повертає список проектів користувача
GET	/projects/:id	повертає повні дані про проект за ідентифікаційним номером проекту
POST	/projects	створює новий проект
POST	/projects/:id/save	зберігає дані про проект
DELETE	/projects/:id	видаляє проект за ідентифікаційним номером проекту

В якості системи управління базами даних використовується Postgres. База даних серверу призначена для зберігання користувачів, а також даних про їхні проекти. Опис таблиць бази даних наведено у таблицях 2.11 - 2.14. Діаграма бази даних наведена у додатках.

Таблиця 2.11 – Опис таблиці user

Таблиця	Назва стовпчика	Тип даних	Опис
user	id	serial	ідентифікаційний номер користувача
	email	varchar	електронна пошта користувача
	password_id	int	посилання на запис у таблиці password, де зберігається пароль користувача

Таблиця 2.12 – Опис таблиці password

Таблиця	Назва стовпчика	Тип даних	Опис
password	id	serial	ідентифікаційний номер даних про пароль користувача
	hash	varchar	хеш паролю користувача, зроблений за останньою схемою хешування, можливий випадок коли у полі зберігається вкладений хеш паролю
	salt	varchar	випадковий рядок, який додається до паролю перед хешуванням для того, щоб уникнути ситуації коли однакові паролі мають однакові хеші
	version	int	версія хешування застосована до паролю
	compromised	boolean	флаг, який показує чи не скомпрометований запис, може використовуватись для того щоб написати код для забезпеченню мір з безпеки на випадок витоку з бази даних завчасно

Продовження таблиці 2.12

Таблиця	Назва стовпчика	Тип даних	Опис
password	updatedTo	int	показує до якої версії хешування був тимчасово оновлений пароль

Таблиця 2.13 – Опис таблиці project

Таблиця	Назва стовпчика	Тип даних	Опис
project	id	serial	ідентифікаційний номер проекту
	user_id	int	ідентифікаційний номер користувача якому належить проект, посилання на запис з таблиці user
	name	varchar	назва проекту
	updatedAt	timestamp	дата і час останнього зберігання проекту
	createdAt	timestamp	дата і час створення проекту
	fieldsJson	text	дані про поля конфігурації проекту у форматі JSON, так як збереження відбувається рідко, а поля можуть часто змінюватись

Таблиця 2.14 – Опис таблиці entity

Таблиця	Назва стовпчика	Тип даних	Опис
project	id	int	ідентифікаційний номер елементу
	project_id	int	ідентифікаційний номер проекту до якого відноситься елемент, посилання на запис з таблиці project
	x	int	координата x елементу на робочій області
	y	int	координата y елементу на робочій області
	type	varchar	тип елементу (мікросервіс, база даних, брокер повідомлень, API шлюз)
	fieldsJson	text	дані про поля конфігурації елементу у форматі JSON, так як збереження відбувається рідко, а поля можуть часто змінюватись

Висновки до розділу

В другому розділі дипломного проекту було розглянуто:

– структуру конфігураційного файлу на основі якого генерується макет застосунку;

– процес роботи консольного застосунку, який генерує макет мікросервісного застосунку;

– набір декораторів, який забезпечує легке додавання нових елементів у застосунок;

– набір модулів, компонентів і сервісів, з яких складається клієнтська частина проекту;

– набір модулів, контролерів і провайдерів з яких складається серверна частина проекту;

– залежності консольного додатку, клієнтської і серверної частини проекту;

– структура бази даних і її таблиць.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості

Під час аналізу якості програмне забезпечення було перевірено на відповідність нефункціональним та функціональним вимогам. Результати перевірки для нефункціональних вимог наведені у таблиці 3.1. Перевірка функціональних вимог була виконана за допомогою ручного тестування. Відповідність між функціональними вимогами і тестами, в яких перевіряється їх виконання, наведено у таблиці 3.2.

Таблиця 3.1 – Перевірка нефункціональних вимог

Зрозумілий інтерфейс Легкість у використанні	Перевірка інтерфейсу показала, що він є зрозумілим, зручним і естетично приємним.
Сумісність з сучасними браузерами	Робота веб-додатку була перевірена на останніх версіях браузерів Firefox, Chrome і Opera
Захищеність від кібер-атак	Для комунікацій з клієнтом і сервером використовується захищений протокол HTTPS, для зниження ймовірності зламу аккаунтів використовуються досить жорсткі вимоги до складності паролів, для їх хешування використовується argon2
Продуктивність роботи	Середній час відповіді сервера – 280мс. Це дещо більше ніж 200мс, що вважається хорошим результатом, але цим можна знехтувати, так як під час роботи з веб-додатком запити на сервер відбуваються рідко (як правило тільки на початку і в кінці роботи).

Таблиця 3.2 – Відповідність тестів і функціональних вимог

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17	FR-18	FR-19	FR-20	FR-21	FR-22	FR-23	FR-24	FR-25	FR-26
Тест 1.1	+																									
Тест 1.2	+																									
Тест 1.3	+																									
Тест 1.4	+																									
Тест 2.1		+																								
Тест 2.2		+																								
Тест 3.1			+																							
Тест 3.2			+																							
Тест 3.3						+	+	+																		
Тест 3.4				+																						
Тест 3.5				+																						
Тест 3.6					+																					
Тест 3.7					+																					
Тест 3.8						+																				
Тест 3.9									+																	
Тест 3.10									+																	
Тест 4.1										+	+	+	+													
Тест 4.2														+	+	+	+									
Тест 4.3														+	+	+	+									
Тест 4.4																		+								
Тест 4.5																		+								
Тест 4.6																			+							
Тест 4.7																			+							
Тест 4.8																				+	+	+	+			
Тест 4.9																				+	+	+	+			
Тест 4.10																								+		
Тест 4.11																								+		
Тест 5.1										+	+	+	+	+	+	+	+	+			+	+	+	+	+	+

3.2 Опис процесів тестування

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 3.3 – 3.30.

Таблиця 3.3 – Тест 1.1

Тест	Реєстрація користувача
Модуль	Реєстрація користувача
Номер тесту	1.1
Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні данні	Електронна пошта, пароль, підтвердження паролю
Опис проведення тесту	У відповідні поля вводяться: коректна електронна пошта, яка до цього не була зареєстрована в системі, пароль від 10 до 64 символів, який містить хоча б з одну англійську літеру, одне число і один спеціальний символ, і який не входить у топ 10000 найпопулярніших паролей, підтвердження паролю, яке співпадає з раніше введеним паролем. Після цього натискається кнопка підтвердження реєстрації.

Продовження таблиці 3.4

Очікуваний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку авторизації.
Фактичний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку авторизації.

Таблиця 3.4 – Тест 1.2

Тест	Реєстрація користувача з неправильною електронною поштою
Модуль	Реєстрація користувача
Номер тесту	1.2
Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні данні	Електронна пошта, пароль, підтвердження паролю
Опис проведення тесту	У відповідні поля вводяться: некоректна електронна пошта або пошта, яка до цього не була зареєстрована в системі, пароль від 10 до 64 символів, який містить хоча б з одну англійську літеру, одне число і один спеціальний символ, і який не входить у топ 10000 найпопулярніших паролей, підтвердження паролю, яке співпадає з раніше введеним паролем. Після цього натискається кнопка підтвердження реєстрації.
Очікуваний результат	Користувач не додається у систему і отримує повідомлення про некоректну електронну пошту з описом конкретної помилки.
Фактичний результат	Користувач не додається у систему і отримує повідомлення про некоректну електронну пошту з описом конкретної помилки.

Таблиця 3.5 – Тест 1.3

Тест	Реєстрація користувача з неправильним паролем
Модуль	Реєстрація користувача
Номер тесту	1.3
Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні данні	Електронна пошта, пароль, підтвердження паролю
Опис проведення тесту	У відповідні поля вводяться: коректна електронна пошта, яка до цього не була зареєстрована в системі, пароль який не відповідає обмеженням на довжину паролю (менше 10 символів, або більше ніж 64 символи) або не відповідає обмеженням на склад символів (немає ні одної англійської літери, числа або спеціального символу) або знаходиться у списку серед 10000 найпопулярніших паролей, підтвердження паролю, яке співпадає з раніше введеним паролем. Після цього натискається кнопка підтвердження реєстрації.
Очікуваний результат	Користувач не додається у систему і отримує повідомлення про некоректний пароль з описом конкретної помилки.
Фактичний результат	Користувач не додається у систему і отримує повідомлення про некоректний пароль з описом конкретної помилки.

Таблиця 3.6 – Тест 1.4

Тест	Реєстрація користувача з неправильним підтвердженням паролю
Модуль	Реєстрація користувача
Номер тесту	1.4
Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні данні	Електронна пошта, пароль, підтвердження паролю
Опис проведення тесту	У відповідні поля вводяться: коректна електронна пошта, яка до цього не була зареєстрована в системі, пароль від 10 до 64 символів, який містить хоча б з одну англійську літеру, одне число і один спеціальний символ, і який не входить у топ 10000 найпопулярніших паролей, підтвердження паролю, яке не співпадає з раніше введеним паролем. Після цього натискається кнопка підтвердження реєстрації.
Очікуваний результат	Користувач не додається у систему і отримує повідомлення про те, що поля пароля і підтвердження пароля не співпадають.
Фактичний результат	Користувач не додається у систему і отримує повідомлення про те, що поля пароля і підтвердження пароля не співпадають.

Таблиця 3.7 – Тест 2.1

Тест	Авторизація користувача
Номер тесту	2.1
Модуль	Авторизація користувача
Початковий стан системи	Користувач знаходиться на сторінці авторизації, перед цим зареєструвавшись
Вхідні данні	Електронна пошта, пароль
Опис проведення тесту	У відповідні поля вводяться: коректна електронна пошта і пароль, які до цього були вказані при реєстрації. Після цього натискається кнопка підтвердження авторизації.
Очікуваний результат	Користувач авторизується у системі і перенаправляється на сторінку управління проектами.
Фактичний результат	Користувач авторизується у системі і перенаправляється на сторінку управління проектами.

Таблиця 3.8 – Тест 2.2

Тест	Невдала авторизація користувача
Номер тесту	2.2
Модуль	Авторизація користувача
Початковий стан системи	Користувач знаходиться на сторінці авторизації, перед цим зареєструвавшись
Вхідні данні	Електронна пошта, пароль
Опис проведення тесту	У відповідні поля вводяться: коректна електронна пошта і пароль, не вказаний при реєстрації. Після цього натискається кнопка підтвердження авторизації.
Очікуваний результат	Користувач отримує повідомлення про те, що комбінація електронної пошти і паролю є некоректною.
Фактичний результат	Користувач отримує повідомлення про те, що комбінація електронної пошти і паролю є некоректною.

Таблиця 3.9 – Тест 3.1

Тест	Перевірка списку проектів
Модуль	Управління проектами
Номер тесту	3.1
Початковий стан системи	Користувач є зареєстрованим і авторизованим, перед цим їм було створено декілька проектів
Вхідні данні	-
Опис проведення тесту	Відбувається вхід на сторінку управління проектами.
Очікуваний результат	Користувач бачить список раніше створених проектів, при натисканні на будь-який з них, відображається назва проекту, дата створення та останньої зміни проекту, кнопка для завантаження проекту.
Фактичний результат	Користувач бачить список раніше створених проектів, при натисканні на будь-який з них, відображається назва проекту, дата створення та останньої зміни проекту, кнопка для завантаження проекту.

Таблиця 3.10 – Тест 3.2

Тест	Створення нового проекту
Модуль	Управління проектами
Номер тесту	3.2
Початковий стан системи	Користувач є зареєстрованим і авторизованим
Вхідні данні	-
Опис проведення тесту	Відбувається вхід на сторінку управління проектами та натискається кнопка створення нового проекту.

Продовження таблиці 3.10

Очікуваний результат	Користувач перенаправляється на сторінку конструювання проекту з пустим проектом, який може бути відредагований і збережений.
Фактичний результат	Користувач перенаправляється на сторінку конструювання проекту з пустим проектом, який може бути відредагований і збережений.

Таблиця 3.11 – Тест 3.3

Тест	Збереження і завантаження проекту
Модуль	Управління проектами
Номер тесту	3.3
Початковий стан системи	Користувач є зареєстрованим і авторизованим, і знаходиться на сторінці конструювання проекту
Вхідні данні	-
Опис проведення тесту	Натискається кнопка збереження проекту. Відбувається перенаправлення на сторінку управління проектами. Збережений проект обирається зі списку. Натискається кнопка завантаження проекту.
Очікуваний результат	Користувач перенаправляється на сторінку конструювання проекту з відкритим проектом, при цьому всі налаштування проекту, створені елементи і зв’язки між ними відповідають тим, що були перед збереженням проекту.
Фактичний результат	Користувач перенаправляється на сторінку конструювання проекту з відкритим проектом, при цьому всі налаштування проекту, створені елементи і зв’язки між ними відповідають тим, що були перед збереженням проекту.

Таблиця 3.12 – Тест 3.4

Тест	Зміна назви проекту
Модуль	Управління проектами
Номер тесту	3.4
Початковий стан системи	Користувач є зареєстрованим і авторизованим і знаходиться на сторінці конструювання проекту
Вхідні данні	Назва проекту
Опис проведення тесту	Натискається поле назви проекту та вводиться нова назва (яка не є порожнім рядком).
Очікуваний результат	Назва не змінюється після зміщення фокусу на інший елемент сторінки.
Фактичний результат	Назва не змінюється після зміщення фокусу на інший елемент сторінки.

Таблиця 3.13 – Тест 3.5

Тест	Некоректна зміна назви проекту
Модуль	Управління проектами
Номер тесту	3.5
Початковий стан системи	Користувач є зареєстрованим і авторизованим і знаходиться на сторінці конструювання проекту
Вхідні данні	Назва проекту
Опис проведення тесту	Натискається поле назви проекту. У нього вводиться порожній рядок.
Очікуваний результат	Назва залишається пустою після зміщення фокусу на інший елемент сторінки, на панелі помилок з’являється повідомлення про те, що назва не може бути пустою.
Фактичний результат	Назва залишається пустою після зміщення фокусу на інший елемент сторінки, на панелі помилок з’являється повідомлення про те, що назва не може бути пустою.

Таблиця 3.14 – Тест 3.6

Тест	Зміна конфігурації проекту
Модуль	Управління проектами
Номер тесту	3.6
Початковий стан системи	Користувач є зареєстрованим і авторизованим і знаходиться на сторінці конструювання проекту
Вхідні данні	Платформа для CI/CD, ім'я користувача у DockerHub
Опис проведення тесту	Натискається кнопка редагування параметрів проекту, з’являється панель редагування конфігурації проекту. Там обирається платформа для CI/CD, і вводиться ім’я користувача DockerHub.
Очікуваний результат	Конфігурація проекту зберігається.
Фактичний результат	Конфігурація проекту зберігається.

Таблиця 3.15 – Тест 3.7

Тест	Некоректна зміна конфігурації проекту
Модуль	Управління проектами
Номер тесту	3.7
Початковий стан системи	Користувач є зареєстрованим і авторизованим і знаходиться на сторінці конструювання проекту
Вхідні данні	Платформа для CI/CD, ім'я користувача у DockerHub
Опис проведення тесту	Натискається кнопка редагування параметрів проекту, з’являється панель редагування конфігурації проекту. Обирається платформа для CI/CD. У поле вводу ім’я користувача DockerHub вводиться пустий рядок.

Продовження таблиці 3.15

Очікуваний результат	Конфігурація проекту зберігається, на панелі помилок з’являється повідомлення про те, що ім’я користувача DockerHub не може бути пустим.
Фактичний результат	Конфігурація проекту зберігається, на панелі помилок з’являється повідомлення про те, що ім’я користувача DockerHub не може бути пустим.

Таблиця 3.16 – Тест 3.8

Тест	Невдале збереження проекту
Модуль	Управління проектами
Номер тесту	3.8
Початковий стан системи	Користувач є зареєстрованим і авторизованим і знаходиться на сторінці конструювання проекту. На панелі помилок знаходиться хоча б одне повідомлення про помилку.
Вхідні данні	-
Опис проведення тесту	Натискається кнопка збереження проекту.
Очікуваний результат	Проект не зберігається, користувач отримує повідомлення про те, що вказує на те, що збереження за наявності помилок неможливе.
Фактичний результат	Проект не зберігається, користувач отримує повідомлення про те, що вказує на те, що збереження за наявності помилок неможливе.

Таблиця 3.17 – Тест 3.9

Тест	Видалення проекту
Модуль	Управління проектами
Номер тесту	3.9
Початковий стан системи	Користувач є зареєстрованим і авторизованим, користувачем було створено хоча б один проект
Вхідні данні	-
Опис проведення тесту	Зі списку обирається проект. Натискається кнопка видалення. Відкривається діалог, який просить підтвердження бажання видалити проект. Натискається кнопка підтвердження видалення.
Очікуваний результат	Проект видаляється з системи, та зникає зі списку проектів.
Фактичний результат	Проект видаляється з системи, та зникає зі списку проектів.

Таблиця 3.18 – Тест 3.10

Тест	Відміна видалення проекту
Модуль	Управління проектами
Номер тесту	3.10
Початковий стан системи	Користувач є зареєстрованим і авторизованим, користувачем було створено хоча б один проект
Вхідні данні	-
Опис проведення тесту	Зі списку обирається проект. Натискається кнопка видалення. Відкривається діалог, який просить підтвердження бажання видалити проект. Натискається кнопка відміни видалення.
Очікуваний результат	Проект не видаляється з системи.
Фактичний результат	Проект не видаляється з системи.

Таблиця 3.19 – Тест 4.1

Тест	Створення елементу
Модуль	Управління елементами
Номер тесту	4.1
Початковий стан системи	Користувач є зареєстрованим і авторизованим, користувач знаходиться на сторінці конструювання проекту
Вхідні данні	-
Опис проведення тесту	Обирається елемент на панелі інструментів зі списку і двічі натискається.
Очікуваний результат	Елемент з’являється у робочій області.
Фактичний результат	Елемент з’являється у робочій області.

Таблиця 3.20 – Тест 4.2

Тест	Зміна налаштувань елементу
Модуль	Управління елементами
Номер тесту	4.2
Початковий стан системи	Користувач є зареєстрованим і авторизованим, користувач знаходиться на сторінці конструювання проекту, їм був створений хоча б один елемент
Вхідні данні	-
Опис проведення тесту	На робочій області обирається елемент і натискається двічі. Відкривається панель редагування налаштувань елементу з відповідними полями в залежності від типу елемент (сервіс, база даних, API шлюз, брокер повідомлень). Вводяться непусті значення.
Очікуваний результат	Значення зберігаються.
Фактичний результат	Значення зберігаються.

Таблиця 3.21 – Тест 4.3

Тест	Некоректна зміна налаштувань елементу
Модуль	Управління елементами
Номер тесту	4.3
Початковий стан системи	Користувач є зареєстрованим і авторизованим, користувач знаходиться на сторінці конструювання проекту, їм був створений хоча б один елемент
Вхідні данні	-
Опис проведення тесту	На робочій області обирається елемент і натискається двічі. Відкривається панель редагування налаштувань елементу з відповідними полями в залежності від типу елемент (сервіс, база даних, API шлюз, брокер повідомлень). Вводяться значення, при цьому в деякі поля вводяться пусті рядки.
Очікуваний результат	Значення зберігаються, на панелі помилок з’являється повідомлення про те, що відповідне поле не може бути пустим.
Фактичний результат	Значення зберігаються, на панелі помилок з’являється повідомлення про те, що відповідне поле не може бути пустим.

Таблиця 3.22 – Тест 4.4

Тест	Зміна розташування елементу
Модуль	Управління елементами
Номер тесту	4.4
Початковий стан системи	Користувач є зареєстрованим і авторизованим, користувач знаходиться на сторінці конструювання проекту, їм був створений хоча б один елемент
Вхідні данні	-

Продовження таблиці 3.22

Опис проведення тесту	На робочій області обирається елемент і перетягується за верхню частину у певне місце всередині робочої області.
Очікуваний результат	Елемент змінює своє розташування, лінії, що показують зв’язки між елементами, також оновлюють своє розташування відповідно.
Фактичний результат	Елемент змінює своє розташування, лінії, що показують зв’язки між елементами, також оновлюють своє розташування відповідно.

Таблиця 3.23 – Тест 4.5

Тест	Некоректна зміна розташування елементу
Модуль	Управління елементами
Номер тесту	4.5
Початковий стан системи	Користувач є зареєстрованим і авторизованим, користувач знаходиться на сторінці конструювання проекту, їм був створений хоча б один елемент
Вхідні данні	-
Опис проведення тесту	На робочій області обирається елемент і перетягується за межі робочої області.
Очікуваний результат	Елемент змінює своє розташування, лінії, що показують зв’язки між елементами, також оновлюють своє розташування відповідно, при контакті з межею робочої області елемент зупиняється.
Фактичний результат	Елемент змінює своє розташування, лінії, що показують зв’язки між елементами, також оновлюють своє розташування відповідно, при контакті з межею робочої області елемент зупиняється.

Таблиця 3.24 – Тест 4.6

Тест	Видалення елементу
Модуль	Управління елементами
Номер тесту	4.6
Початковий стан системи	Користувач є зареєстрованим і авторизованим, користувач знаходиться на сторінці конструювання проекту, їм був створений хоча б один елемент
Вхідні данні	-
Опис проведення тесту	На робочій області обирається елемент і натискається двічі. Відкривається панель редагування налаштувань елементу. Натискається кнопка видалення елементу. Відкривається діалог, який просить підтвердження бажання видалити елемент. Натискається кнопка підтвердження видалення.
Очікуваний результат	Елемент і всі його зв’язки зникають з робочої області.
Фактичний результат	Елемент і всі його зв’язки зникають з робочої області.

Таблиця 3.25 – Тест 4.7

Тест	Відміна видалення елементу
Модуль	Управління елементами
Номер тесту	4.7
Початковий стан системи	Користувач є зареєстрованим і авторизованим, користувач знаходиться на сторінці конструювання проекту, їм був створений хоча б один елемент
Вхідні данні	-

Продовження таблиці 3.26

Опис проведення тесту	На робочій області обирається елемент і натискається двічі. Відкривається панель редагування налаштувань елементу. Натискається кнопка видалення елементу. Відкривається діалог, який просить підтвердження бажання видалити елемент. Натискається кнопка відміни видалення.
Очікуваний результат	Елемент залишається на робочій області.
Фактичний результат	Елемент залишається на робочій області.

Таблиця 3.26 – Тест 4.8

Тест	Створення зв’язку між елементами
Модуль	Управління елементами
Номер тесту	4.8
Початковий стан системи	Користувач є зареєстрованим і авторизованим, користувач знаходиться на сторінці конструювання проекту, їм був створений хоча б два елементи
Вхідні данні	-
Опис проведення тесту	На робочій області обирається та натискається елемент, курсор перетягується до іншого елементу і відпускається. При цьому елементи є сумісними (наприклад, база даних і мікросервіс).
Очікуваний результат	З’являється лінія, що відображає зв’язок між елементами.
Фактичний результат	З’являється лінія, що відображає зв’язок між елементами.

Таблиця 3.27 – Тест 4.9

Тест	Невдале створення зв’язку між елементами
Модуль	Управління елементами
Номер тесту	4.9
Початковий стан системи	Користувач є зареєстрованим і авторизованим, користувач знаходиться на сторінці конструювання проекту, їм був створений хоча б два елементи
Вхідні данні	-
Опис проведення тесту	На робочій області обирається та натискається елемент, курсор перетягується до іншого елементу і відпускається. При цьому елементи є несумісними (наприклад, база даних і API шлюз).
Очікуваний результат	На панелі помилок з’являється повідомлення про те, що елементи не є сумісними.
Фактичний результат	На панелі помилок з’являється повідомлення про те, що елементи не є сумісними.

Таблиця 3.28 – Тест 4.10

Тест	Видалення зв’язків елементу
Модуль	Управління елементами
Номер тесту	4.10
Початковий стан системи	Користувач є зареєстрованим і авторизованим, користувач знаходиться на сторінці конструювання проекту, їм був створений і пов’язані хоча б два елементи
Вхідні данні	-

Продовження таблиці 3.28

Опис проведення тесту	На робочій області обирається елемент і натискається двічі. Відкривається панель редагування налаштувань елементу. Натискається кнопка видалення усіх зв’язків елементу. Відкривається діалог, який просить підтвердження бажання видалити усі зв’язки елементу. Натискається кнопка підтвердження видалення.
Очікуваний результат	Всі зв’язки елементу видаляються.
Фактичний результат	Всі зв’язки елементу видаляються.

Таблиця 3.29 – Тест 4.11

Тест	Відміна видалення зв’язків елементу
Модуль	Управління елементами
Номер тесту	4.10
Початковий стан системи	Користувач є зареєстрованим і авторизованим, користувач знаходиться на сторінці конструювання проекту, їм був створений і пов’язані хоча б два елементи
Вхідні данні	-
Опис проведення тесту	На робочій області обирається елемент і натискається двічі. Відкривається панель редагування налаштувань елементу. Натискається кнопка видалення усіх зв’язків елементу. Відкривається діалог, який просить підтвердження бажання видалити усі зв’язки елементу. Натискається кнопка відміни видалення.
Очікуваний результат	Всі зв’язки елементу не видаляються.
Фактичний результат	Всі зв’язки елементу не видаляються.

Таблиця 3.30 – Тест 5.1

Тест	Генерація макету застосунку
Модуль	Генерація макету застосунку
Номер тесту	5.1
Початковий стан системи	Користувач є зареєстрованим і авторизованим, користувач знаходиться на сторінці конструювання проекту
Вхідні данні	-
Опис проведення тесту	Створюються: API шлюз, 3 мікросервіси і база даних для кожного з них, брокер повідомлень. Для API шлюзу вказується назва хоста. Для кожного мікросервісу вказується назва, порт, кількість реплік, редагується конфігурація API, в якості типу серверу обирається typescript/nestjs. Для кожної бази даних вказується назва, розмір сховища, в якості типу обирається PostgreSQL. Для брокера повідомлень обирається кількість реплік, в якості типу обирається Kafka. Кожен мікросервіс пов’язується зі своєю базою даних. Також всі мікросервіси пов’язуються з API шлюзом і брокером повідомлень. Для проекту в якості CI/CD системи обирається github-actions, вводить ім'я користувача DockerHub. Натискається кнопка завантаження конфігураційного файлу. Запускається CLI, вказується розташування конфігураційного файлу та місце збереження готового макету застосунку. За запитом програми вводяться ключ до GitHub CLI, паролі для всіх баз даних.

Продовження таблиці 3.30

Очікуваний результат	Була згенерована частина коду для кожного мікросервісу, що не включає бізнес-логіку, а саме оголошення серверу, необхідні модулі і контролери з методами що відповідають кінцевим точкам (endpoint) API, DTO (data-transfer-object), файл з описом залежностей серверу, конфігурація лінтеру, Dockerfile тощо. Були згенеровані конфігураційні файли kubernetes для розгортання кожного мікросервісу, кожної бази даних, API шлюзу, брокеру повідомлень. Були згенеровані конфігураційні файли kubernetes, які забезпечують створення постійного сховища для баз даних, та файли, які безпечно зберігають паролі для баз даних.
Фактичний результат	Була згенерована частина коду для кожного мікросервісу, що не включає бізнес-логіку, а саме оголошення серверу, необхідні модулі і контролери з методами що відповідають кінцевим точкам (endpoint) API, DTO (data-transfer-object), файл з описом залежностей серверу, конфігурація лінтеру, Dockerfile тощо. Були згенеровані конфігураційні файли kubernetes для розгортання кожного мікросервісу, кожної бази даних, API шлюзу, брокеру повідомлень. Були згенеровані конфігураційні файли kubernetes, які забезпечують створення постійного сховища для баз даних, та файли, які безпечно зберігають паролі для баз даних.

Висновки до розділу

Було проведено аналіз якості програмного забезпечення, що включало у себе визначення рівня дотримання усіх функціональних і нефункціональних вимог.

Для оцінки виконання нефункціональних вимог було перевірено такі характеристики програмного забезпечення: зрозумілість і зручність інтерфейсу, сумісність з сучасними браузерами, продуктивність роботи на сучасних браузерах, захищеність від кібер-атак. Оцінка цих характеристик показала задовільний рівень відповідності нефункціональним вимогам.

Виконання функціональних вимог перевірялось за допомогою мануального тестування. Кожна функціональна вимога має один чи більше тестів, в яких перевіряється виконання вимоги. Всього тестів – 28. Помилки знайдені під час тестування були виправлені.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Клієнтську і серверну частини програмного забезпечення було вирішено розгорнути на платформі Heroku. Для розгортання було використано сервіс GitHub Actions, який надає можливості для постійної інтеграції і розгортання [13].

Розгортання починається коли новий код застосунку доставляється у репозиторій у гілку main. Тоді у середовищі GitHub Actions створюється Docker image за допомогою Dockerfile, що знаходиться у проекті. Цей image розгортається у Heroku за допомогою пакету heroku-deploy. Інформацію про розгортання клієнтської і серверної частини проекту можна побачити на рисунках 4.1 і 4.2.

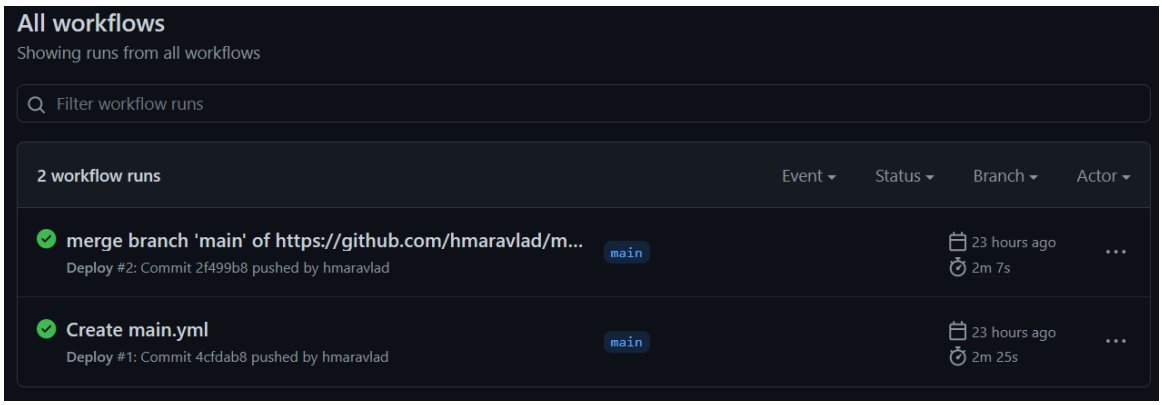


Рисунок 4.1 - Інформація про розгортання клієнту

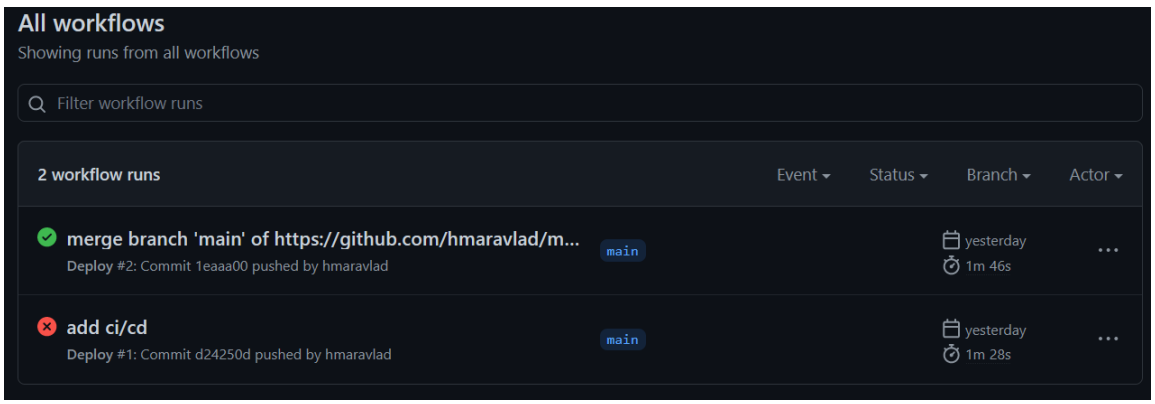


Рисунок 4.2 - Інформація про розгортання серверу

4.2 Випуск програмного забезпечення

Користувачі повинні мати можливість отримати нову версію консольного застосунку з кожною версією. До того ж кожна нова версія консольного застосунку повинна бути опублікована в npm. Для автоматизації цього процесу був використаний сервіс GitHub Actions.

Створення нового випуску починається, коли нова версія консольного застосунку доставляється у репозиторій у гілку main, тобто коли commit має tag формату “v*.*.*.”, де замість “*” знаходиться число. Тоді у середовищі GitHub Actions встановлюється NodeJS. Після цього для проекту встановлюються залежності і проект збирається. Bash скрипт за допомогою бібліотеки pkg генерує виконувані файли (executables) для Linux і для Windows, та пакує файл для Linux у .deb пакет. Після цього .deb пакет і файл для Windows архівуються, створюється новий реліз і ці архіви до нього додаються в якості assets. Також нова версія публікується у npm.

Після цього застосунок потрібної версії можна скачати на сторінці відповідного релізу у GitHub. Сторінку релізів наведено у рисунку 4.3.

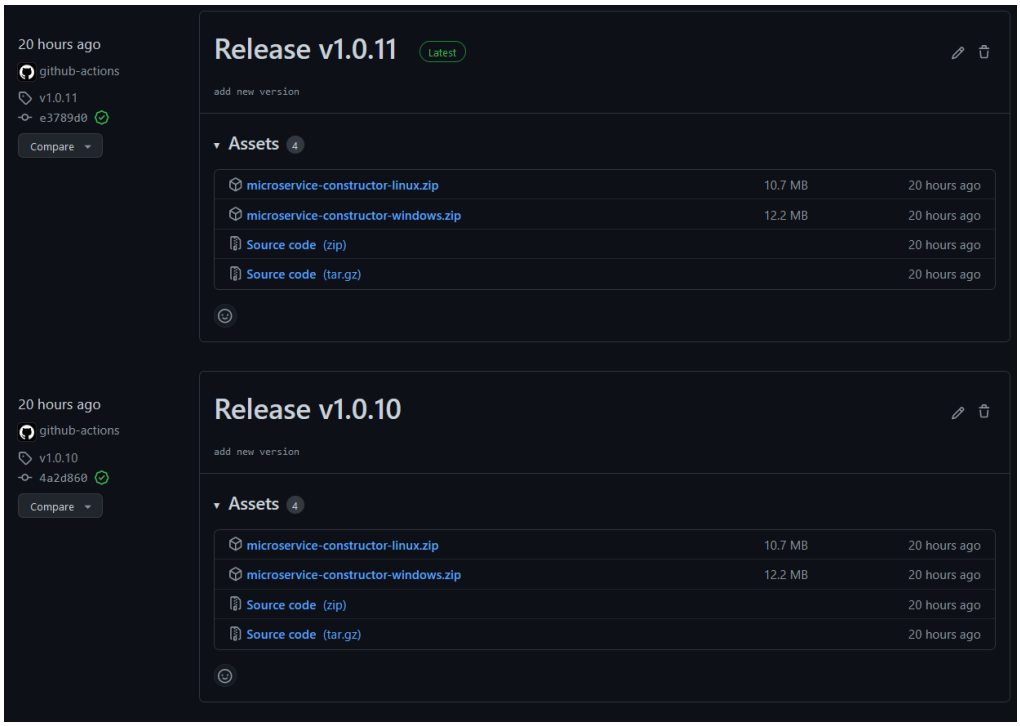


Рисунок 4.3 – Сторінка релізів

Висновки до розділу

Для розгортання і випуску програмного забезпечення було використано підхід CI/CD. В якості сервісу, який надає можливості використання практик CI/CD, було використано GitHub Actions.

За допомогою GitHub Actions було автоматизовано розгортання клієнтської і серверної частини проекту на платформі Heroku.

Також було автоматизовано випуск нових версій консольного додатку, а саме створення виконуваних файлів для Windows і Linux, створення .deb пакету для Linux, публікація у npm, створення релізу у GitHub і додавання до нього архівів з виконуваними файлами.

ВИСНОВКИ

Було вивчено і проаналізовано мікросервісну архітектуру та елементи, які використовуються при її реалізації. Аналіз предметної області і аналогів проекту дозволив розробити функціональні вимоги.

В результаті виконання дипломного проекту було спроектовано програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою.

Програмне забезпечення складається з веб-додатку, який має клієнтську і серверну частину, та консольного додатку. Всі частини програмного забезпечення були написані мовою Typescript. В якості середовища розробки використовувався редактор Visual Studio Code. Веб-додаток надає користувачу графічний інтерфейс для опису архітектури застосунку. Консольний застосунок дозволяє на основі цього опису згенерувати макет застосунку, тобто конфігураційні файли для розгортання сервісів, баз даних, API шлюзів і брокерів повідомлень, а також певну частина коду власне мікросервісів.

Після реалізації веб-застосунку він був протестований на останніх версіях браузерів Firefox, Chrome і Opera. Консольний застосунок було протестовано на пристроях з Ubuntu і Windows 10.

Були виконані такі завдання:

– спрощення і пришвидшення розробки застосунку з мікросервісною архітектурою, за допомогою надання користувачу графічного інтерфейсу, який дозволяє згенерувати макет веб-застосунку;

– зменшення витрат необхідних на початкових етапах розробки, так як програмне забезпечення для конструювання макетів веб-застосунків використовується саме на початку розробки;

– зменшення ймовірності помилок при розробці застосунку з мікросервісною архітектурою, так як конфігураційні файли генеруються автоматично.

Таким чином програмне забезпечення може бути використано розробниками для розробки веб-застосунків з мікросервісною архітектурою. Програмне забезпечення може бути покращено додаванням нових видів елементів і нових варіантів вже доданих елементів (більше варіантів СУБД, серверів тощо).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Brown S. Software Architecture for Developers. Vol 1. Victoria: Leanpub, 2018. 95 с.

2) Mark Richards. Software Architecture Patterns. Sebastopol: O'Reilly, 2015. 47 с.

3) Surianarayanan Ch., Ganapathy G., Raj P. Essentials of Microservices. Architecture Paradigms, Applications, and Techniques. Boca Raton: CRC Press, 2020. 293 с.

4) Wolff E. Microservices. Flexible Software Architecture. Crawfordsville: Pearson Education, 2016. 395 с.

5) Nadareishvili I., Mitra R., McLarty M., Amundsen M. Microservice Architecture. Aligning Principles, Practices and Culture. Sebastopol: O'Reilly, 2016. 126 с.

6) Christudas B. Practical Microservices Architectural Patterns. Trivandrum, Kerala: Apress, 2019. 902 с.

7) Vohra D. Kubernetes Microservices with Docker. White Rock: Apress, 2016. 432 с.

8) Freeman A. Pro Angular 9. London: Apress, 2020. 784 с.

9) Офіційна документація Angular. URL: https://angular.io/docs

10) Офіційна документація Vue. URL: https://devdocs.io/vue~1-guide/

11) Офіційна документація React. URL: https://reactjs.org/docs/getting-started.html

12) Офіційна документація NestJs. URL: https://docs.nestjs.com/

13) Офіційна документація GitHub Actions. URL: https://docs.github.com/en/actions

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

UNICHECK

Имя пользователя:
Лісовиченко Олег Іванович

ID проверки:
1011480874

Дата проверки:
07.06.2022 07:40:24 EEST

Тип проверки:
Doc vs Internet + Library

Дата отчета:
07.06.2022 07:45:20 EEST

ID пользователя:
76913

Название файла: **ІП-81_Хмара_ПЗ**

Количество страниц: **86** Количество слов: **11927** Количество символов: **92444** Размер файла: **1.59 MB** ID файла: **1011358348**

1.71%
Совпадения

Наибольшее совпадение: **0.62%** с источником из Библиотеки (ID файла: **1011358337**)

0.44% Источники из Интернета	22	Страница 88
1.71% Источники из Библиотеки	79	Страница 88

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

Програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою

Опис програми

КПІ.ІП-8321.045490.03.13

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Максим ГОЛОВЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Владислав ХМАРА

Київ – 2022

ТЕКСТ ПРОГРАМНОГО КОДУ

Тексти програмного коду

Програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою

(Найменування програми (документа))

(Вид носія даних)

37 арк, 56 Кб

(Обсяг програми, арк., Кб)

Київ – 2022

Файл microservice-constructor/src/types/config/project-config.ts

```
/* eslint-disable @typescript-eslint/ban-types */
import { Database } from './database';
import { EventBus } from './event-bus';
import { Gateway } from './gateway';
import { ServiceConfig } from './service-config';
import { StringField } from '../../decorators/string-field';
import { Entity } from '../../decorators/entity';
import { EnumField } from '../../decorators/enum-field';
import { availableCICDSetups } from '../../cicd-generator/cicd-generator-
factory';

export class ProjectConfig {
  @StringField
  name: string;

  @EnumField(availableCICDSetups)
  cicd: string;

  @StringField
  dockerUsername: string;

  @Entity('gateway', Gateway)
  gateways: Gateway[];

  @Entity('eventBus', EventBus)
  eventBuses: EventBus[];

  @Entity('service', ServiceConfig)
  services: ServiceConfig[];

  @Entity('database', Database)
  databases: Database[];
}
```

Файл microservice-constructor/src/types/config/database.ts

```
import { IdField } from '../../decorators/id-field';
import { availableDatabases } from '../../database-generator/database-generator-
factory';
import { EnumField } from '../../decorators/enum-field';
import { NumberField } from '../../decorators/number-field';
import { StringField } from '../../decorators/string-field';

export class Database {
  @IdField
  id: number;
```

```
  @StringField
  name: string;

  @EnumField(availableDatabases)
  type: string;

  @NumberField
  sizeGb: number;
}
```

Файл microservice-constructor/src/types/config/service-config.ts

```
import { APIConfig } from './api-config';
import { StringField } from '../../decorators/string-field';
import { NumberField } from '../../decorators/number-field';
import { BooleanField } from '../../decorators/boolean-field';
import { ReferenceField } from '../../decorators/reference-field';
import { CustomTypeField } from '../../decorators/custom-field';
import { EnumField } from '../../decorators/enum-field';
import { availableServiceSetups } from '../../server-generator/server-generator-
factory';
import { IdField } from '../../decorators/id-field';

export class ServiceConfig {
  @IdField
  id: number;

  @StringField
  name: string;

  @EnumField(availableServiceSetups)
  lang: string;

  @NumberField
  port: number;

  @BooleanField
  docs: boolean;

  @NumberField
  replicas: number;

  @ReferenceField(['database'])
  databaseIds: number[];

  @ReferenceField(['eventBus'])
  eventBusIds: number[];

  @CustomTypeField('APIConfig')
```

```
  api?: APIConfig;
}
```

Файл microservice-constructor/src/types/config/gateway.ts

```
import { ReferenceField } from '../../decorators/reference-field';
import { StringField } from '../../decorators/string-field';
import { IdField } from '../../decorators/id-field';

export class Gateway {
  @IdField
  id: number;

  @ReferenceField(['service'])
  serviceIds: number[];

  @StringField
  hostname: string;
}
```

Файл microservice-constructor/src/types/config/event-bus.ts

```
import { EnumField } from '../../decorators/enum-field';
import { availableEventBuses } from '../../event-bus-generator/event-bus-
generator-factory';
import { NumberField } from '../../decorators/number-field';
import { IdField } from '../../decorators/id-field';

export class EventBus {
  @IdField
  id: number;

  @EnumField(availableEventBuses)
  type: string;

  @NumberField
  replicas: number;
}
```

Файл microservice-constructor/src/types/generator-mapping.ts

```
import { SecretsCreator } from '../secret-creator';
import { FilesGenerator } from './files-generator';
import { ProjectConfig } from './config/project-config';

export type GeneratorMapping<T, K> = {
  [key: string]: {
```

```
    getGenerator: (secretsCreator: SecretsCreator, projectConfig: ProjectConfig)
=> FilesGenerator<T>,
    getInfoProvider?: (projectConfig: ProjectConfig) => K
  }
};
```

Файл microservice-constructor/src/decorators/reference-field.ts

```
/* eslint-disable @typescript-eslint/ban-types */

export const ReferenceField = (references: string[]): PropertyDecorator =>
(target: Object, propertyKey: string | symbol) => {
  const metadata = Reflect.getMetadata('fieldData', target) || [];
  metadata.push({ name: propertyKey, type: 'number', references });
  Reflect.defineMetadata('fieldData', metadata, target);
};
```

Файл microservice-constructor/src/decorators/entity.ts

```
/* eslint-disable @typescript-eslint/ban-types */

export const Entity = (name: string, type: Object & { prototype: Object; }):
PropertyDecorator => (target: Object, propertyKey: string | symbol) => {
  const constructor = target;
  const metadata = Reflect.getMetadata('entities', constructor) || [];
  const fields = Reflect.getMetadata('fieldData', type.prototype);
  metadata.push({
    name,
    fieldName: propertyKey,
    fields,
  });
  Reflect.defineMetadata('entities', metadata, constructor);
};
```

Файл microservice-constructor/src/decorators/boolean-field.ts

```
/* eslint-disable @typescript-eslint/ban-types */

export const BooleanField: PropertyDecorator = (target: Object, propertyKey:
string | symbol) => {
  const metadata = Reflect.getMetadata('fieldData', target) || [];
  metadata.push({ name: propertyKey, type: 'boolean' });
  Reflect.defineMetadata('fieldData', metadata, target);
};
```

Файл microservice-constructor/src/decorators/string-field.ts

```
/* eslint-disable @typescript-eslint/ban-types */
```

```
export const StringField: PropertyDecorator = (target: Object, propertyKey:
string | symbol) => {
  const metadata = Reflect.getMetadata('fieldData', target) || [];
  metadata.push({ name: propertyKey, type: 'string' });
  Reflect.defineMetadata('fieldData', metadata, target);
};
```

Файл microservice-constructor/src/decorators/custom-field.ts

```
/* eslint-disable @typescript-eslint/ban-types */

export const CustomTypeField = (type: string): PropertyDecorator => (target:
Object, propertyKey: string | symbol) => {
  const metadata = Reflect.getMetadata('fieldData', target) || [];
  metadata.push({ name: propertyKey, type });
  Reflect.defineMetadata('fieldData', metadata, target);
};
```

Файл microservice-constructor/src/decorators/number-field.ts

```
/* eslint-disable @typescript-eslint/ban-types */

export const NumberField: PropertyDecorator = (target: Object, propertyKey:
string | symbol) => {
  const metadata = Reflect.getMetadata('fieldData', target) || [];
  metadata.push({ name: propertyKey, type: 'number' });
  Reflect.defineMetadata('fieldData', metadata, target);
};
```

Файл microservice-constructor/src/decorators/id-field.ts

```
/* eslint-disable @typescript-eslint/ban-types */

export const IdField: PropertyDecorator = (target: Object, propertyKey: string |
symbol) => {
  const metadata = Reflect.getMetadata('fieldData', target) || [];
  metadata.push({ name: propertyKey, type: 'number', isId: true });
  Reflect.defineMetadata('fieldData', metadata, target);
};
```

Файл microservice-constructor/src/decorators/enum-field.ts

```
/* eslint-disable @typescript-eslint/ban-types */

export const EnumField = (possibleValues: string[] | number[]): PropertyDecorator
=> (target: Object, propertyKey: string | symbol) => {
```

```
  if (possibleValues.length === 0) throw new Error('There should be at least one
possible value');
  const metadata = Reflect.getMetadata('fieldData', target) || [];
  metadata.push({ name: propertyKey, type: 'enum', possibleValues });
  Reflect.defineMetadata('fieldData', metadata, target);
};
```

Файл microservice-constructor/src/main.ts

```
import { readFileSync } from 'fs';
import { parse } from 'ts-command-line-args';
import { generateProject } from './project-generator';
import { IArgs, argsConfig } from './types/command-line-args';
import { ProjectConfig } from './types/config/project-config';
import { FileWriter } from './writer';
import 'reflect-metadata';

(async () => {
  const args = parse<IArgs>(argsConfig);
  const data = readFileSync(args.source).toString();
  const config: ProjectConfig = JSON.parse(data);
  const files = await generateProject(config);
  const writer = new FileWriter(args.target);
  await writer.writeFiles(files);
})();
```

Файл microservice-constructor/src/gateway-generator/templates/ingress-srv.template.ts

```
import { getDistinct } from '../../utils/distinct';
import { File } from '../../types/file';
import { FileTemplate } from '../../types/file-template';
import { ProjectConfig } from '../../types/config/project-config';
import { removeEmptyLines } from '../../utils/remove-empty-lines';
import { Gateway } from '../../types/config/gateway';

export class IngressSrvTemplate implements FileTemplate<Gateway> {
  constructor(private config: ProjectConfig) {}

  getFile(gateway: Gateway): File {
    if (!this.checkPrefixes(this.config, gateway)) throw new Error('Prefixes must
be unique');

    return {
      name: `ingress-${gateway.id}-srv.yaml`,
      path: 'infra/',
      data: `
        apiVersion: extensions/v1beta1
```

```
        kind: Ingress
        metadata:
          name: ingress-service
          annotations:
            kubernetes.io/ingress.class: nginx
            nginx.ingress.kubernetes.io/use-regex: 'true'
        spec:
          rules:
            - host: ${gateway.hostname}
              http:
                paths: ${'\n' +
removeEmptyLines(this.config.services.filter(service =>
gateway.serviceIds.includes(service.id)).flatMap(serviceConfig =>
serviceConfig.api?.endpointGroups.map(endpointGroup => `
                  - path: /${endpointGroup.prefix}/?(.*)
                    backend:
                      serviceName: ${serviceConfig.name}-srv
                      servicePort: ${serviceConfig.port}
                `)).reduce((prev, curr) => (prev || '') + (curr + ''), '') ||
'')}
      `,
    };
  }

  checkPrefixes(config: ProjectConfig, gateway: Gateway): boolean {
    const prefixes = config.services
      .filter(service => gateway.serviceIds.includes(service.id))
      .flatMap(serviceConfig => serviceConfig.api?.endpointGroups.map(group =>
group.prefix));
    return getDistinct(prefixes).length === prefixes.length;
  }
}
```

Файл microservice-constructor/src/gateway-generator/index.ts

```
import { File } from '../types/file';
import { FilesGenerator } from '../types/files-generator';
import { ProjectConfig } from '../types/config/project-config';
import { IngressSrvTemplate } from './templates/ingress-srv.template';

export class GatewayGenerator implements FilesGenerator<ProjectConfig> {
  generateFiles(config: ProjectConfig): File[] {
    const files: File[] = [];
    for (const gateway of config.gateways) {
      files.push(new IngressSrvTemplate(config).getFile(gateway));
    }
    return files;
  }
}
```

Файл microservice-constructor/src/service-k8s-config-generator/templates/depl.template.ts

```
import {  databaseInfoProviderFactory } from '../../database-generator/database-generator-factory';
import { EnvVariable } from '../../types/env-variable';
import { addIndentation } from '../../utils/add-indentation';
import { removeEmptyLines } from '../../utils/remove-empty-lines';
import { removeExtraWhiteSpace } from '../../utils/remove-extra-white-space';
import { File } from '../../types/file';
import { FileTemplate } from '../../types/file-template';
import { ProjectConfig } from '../../types/config/project-config';
import { ServiceConfig } from '../../types/config/service-config';
import { eventBusInfoProviderFactory } from '../../event-bus-generator/event-bus-generator-factory';

export class DeplTemplate implements FileTemplate<ServiceConfig> {
  constructor(private projectConfig: ProjectConfig) {}

  generateEnv(serviceConfig: ServiceConfig): string {
    const envs: EnvVariable[] = [];
    const secrets: string[] = [];

    const dbs = this.projectConfig.databases.filter(({ id }) => serviceConfig.databaseIds.includes(id) );

    for (const db of dbs) {
      const databaseGenerator = databaseInfoProviderFactory.get(db.type, this.projectConfig);
      envs.push(...databaseGenerator.getEnvVariables(db.id));
      secrets.push(...databaseGenerator.getSecrets(db.id));
    }

    const eventBuses = this.projectConfig.eventBuses.filter(({ id }) => serviceConfig.eventBusIds.includes(id) );

    for (const eventBus of eventBuses) {
      const eventBusGenerator = eventBusInfoProviderFactory.get(eventBus.type, this.projectConfig);
      envs.push(...eventBusGenerator.getEnvVariables(eventBus.id));
    }

    const secretStrs = removeExtraWhiteSpace(secrets
      .map(secret => `
        - name: ${secret}
          valueFrom:
            secretKeyRef:
              name: ${secret.toLowerCase().replace(/_/g, '-')}-secret
              key: ${secret}
```

```
      `).join('\n'));

    const envsStrs = removeExtraWhiteSpace(envs
      .map(env => `
        - name: ${env.name}
          value: '${env.value}'
      `).join('\n'));

    if (secretStrs.length + envsStrs.length > 0) {
      return 'env:' + '\n' + addIndentation(secretStrs, '\t') + '\n'
+  addIndentation(envsStrs, '\t');
    } else {
      return '';
    }
  }

  getFile(serviceConfig: ServiceConfig): File {
    return {
      name: `${serviceConfig.name}-depl.yaml`,
      path: 'infra/',
      data: removeEmptyLines(`
        apiVersion: apps/v1
        kind: Deployment
        metadata:
          name: ${serviceConfig.name}-depl
        spec:
          replicas: ${serviceConfig.replicas}
          selector:
            matchLabels:
              app: ${serviceConfig.name}
          template:
            metadata:
              labels:
                app: ${serviceConfig.name}
            spec:
              containers:
                - name: ${serviceConfig.name}
                  image:
${this.projectConfig.dockerUsername}/${serviceConfig.name}
                  ${addIndentation(this.generateEnv(serviceConfig),
'\t\t\t\t\t\t\t\t\t', true)}
        ---
        apiVersion: v1
        kind: Service
        metadata:
          name: ${serviceConfig.name}-srv
        spec:
          selector:
            app: ${serviceConfig.name}
          ports:
```

```
            - name: ${serviceConfig.name}
              protocol: TCP
              port: ${serviceConfig.port}
              targetPort: ${serviceConfig.port}
      `),
    };
  }
}
```

Файл microservice-constructor/src/service-k8s-config-generator/service-k8s-config-generator.ts

```
import { File } from '../types/file';
import { FilesGenerator } from '../types/files-generator';
import { ProjectConfig } from '../types/config/project-config';
import { ServiceConfig } from '../types/config/service-config';
import { DeplTemplate } from './templates/depl.template';

export class ServiceK8SConfigGenerator implements FilesGenerator<ServiceConfig> {
  constructor(private projectConfig: ProjectConfig) {}

  generateFiles(config: ServiceConfig): File[] {
    const deplTemplate = new DeplTemplate(this.projectConfig);
    return [deplTemplate.getFile(config)];
  }
}
```

Файл microservice-constructor/src/info-provider-factory.ts

```
import { GeneratorMapping } from './types/generator-mapping';
import { ProjectConfig } from './types/config/project-config';

export class InfoProviderFactory<T> {
  constructor(public generatorMapping: GeneratorMapping<unknown, T>) {}

  get(type: string, projectConfig: ProjectConfig): T {
    const getInfoProvider = this.generatorMapping[type].getInfoProvider;
    if (!getInfoProvider) {
      throw new Error(`${type} is not supported`);
    }
    return getInfoProvider(projectConfig);
  }
}
```

Файл microservice-constructor/src/utils/get-files-getter.ts

```
/* eslint-disable @typescript-eslint/no-explicit-any */
import { GeneratorFactory } from '../generator-factory';
```

```
import { SecretsCreator } from '../secret-creator';
import { File } from '../types/file';
import { ProjectConfig } from '../types/config/project-config';

const generateFromOneInput = <T extends { [key: string]: any }>(factory:
GeneratorFactory<T>, input: T, typeField: string, secretsCreator: SecretsCreator,
projectConfig: ProjectConfig) => {
  const type = input[typeField];
  if (typeof type !== 'string') throw new Error(`Type field ${typeField} is
invalid`);
  const generator = factory.get(type, secretsCreator, projectConfig);
  return generator.generateFiles(input);
};

export const getFilesGetter = (secretsCreator: SecretsCreator, projectConfig:
ProjectConfig) => <T extends { [key: string]: any }>(factory:
GeneratorFactory<T>, inputData: T | T[], typeField: string): File[] => {
  const files: File[] = [];
  if (Array.isArray(inputData)) {
    for (const input of inputData) {
      files.push(...generateFromOneInput(factory, input, typeField,
secretsCreator, projectConfig));
    }
  } else {
    files.push(...generateFromOneInput(factory, inputData, typeField,
secretsCreator, projectConfig));
  }
  return files;
};
```

Файл microservice-constructor/src/utils/reflection.ts

```
import { ProjectConfig } from '../types/config/project-config';
import { FieldData, EntityData } from '../types/entity';

export function getProjectConfigData(): FieldData[] {
  return Reflect.getMetadata('fieldData', ProjectConfig.prototype);
}

export function getEntitiesData(): EntityData[] {
  return Reflect.getMetadata('entities', ProjectConfig.prototype);
}
```

Файл microservice-constructor/src/utils/remove-empty-lines.ts

```
export function removeEmptyLines(str: string): string {
  return str
    .split('\n')
    .filter(line => line.trim() !== '')
```

```
    .join('\n');
}
```

Файл microservice-constructor/src/utils/add-indentation.ts

```
export function addIndentation(str: string, indent: string,
removeIndentForFirstRow = false): string {
  return str
    .split('\n')
    .map(row => indent + row)
    .join('\n')
    .slice(removeIndentForFirstRow ? indent.length : 0);
}
```

Файл microservice-constructor/src/utils/remove-extra-white-space.ts

```
export function removeExtraWhiteSpace(data: string): string {
  let minNumber = Number.MAX_SAFE_INTEGER;
  const rows = data.split('\n')
    .map(row => row.replace(/\t/g, '  '));
  for (const row of rows) {
    if (row.trim() === '') continue;
    for (let i = 0; i < row.length; i++) {
      if (!row[i].match(/\s/)) {
        minNumber = Math.min(minNumber, i);
        break;
      }
    }
  }
  return rows.map(row => row.slice(minNumber)).join('\n');
}
```

Файл microservice-constructor/src/event-bus-generator/kafka-event-bus-generator/index.ts

```
import { SecretsCreator } from '../../secret-creator';
import { ProjectConfig } from '../../types/config/project-config';
import { EventBus } from '../../types/config/event-bus';
import { File } from '../../types/file';
import { FileTemplate } from '../../types/file-template';
import { FilesGenerator } from '../../types/files-generator';
import { KafkaBrokerTemplate } from './templates/kafka-broker.yaml.template';
import { KafkaServiceTemplate } from './templates/kafka-service.yaml.template';
import { ZookeeperDeplTemplate } from './templates/zookeeper-depl.yaml.template';
import { ZookeeperServiceTemplate } from './templates/zookeeper-
service.yaml.template';
```

```
export default class KafkaEventBusGenerator implements
FilesGenerator<EventBus>  {
  constructor(
    private secretsCreator: SecretsCreator,
    private projectConfig: ProjectConfig,
  ) {}

  templates: FileTemplate<EventBus>[] = [
    new ZookeeperDeplTemplate(),
    new ZookeeperServiceTemplate(),
    new KafkaServiceTemplate(),
    new KafkaBrokerTemplate(),
  ];

  generateFiles(config: EventBus): File[] {
    const files = this.templates.map(template => template.getFile(config));
    return files;
  }
}
```

Файл microservice-constructor/src/event-bus-generator/event-bus-generator-factory.ts

```
import { EventBus } from '../types/config/event-bus';
import KafkaEventBusGenerator from './kafka-event-bus-generator';
import { EnvVariableProvider } from './../types/env-provider';
import { GeneratorMapping } from '../types/generator-mapping';
import { KafkaInfoProvider } from './kafka-event-bus-generator/kafka-info-
provider';
import { ProjectConfig } from '../types/config/project-config';
import { SecretsCreator } from '../secret-creator';
import { InfoProviderFactory } from '../info-provider-factory';
import { GeneratorFactory } from '../generator-factory';

export const eventBusGeneratorMapping: GeneratorMapping<EventBus,
EnvVariableProvider> = {
  'kafka': {
    getGenerator(secretsCreator: SecretsCreator, projectConfig: ProjectConfig) {
      return new KafkaEventBusGenerator(secretsCreator, projectConfig);
    },
    getInfoProvider(projectConfig: ProjectConfig) {
      return new KafkaInfoProvider(projectConfig);
    },
  },
};

export const eventBusGeneratorFactory = new
GeneratorFactory(eventBusGeneratorMapping);
```

```
export const eventBusInfoProviderFactory = new
InfoProviderFactory(eventBusGeneratorMapping as GeneratorMapping<unknown,
EnvVariableProvider>);
export const availableEventBuses = Object.keys(eventBusGeneratorMapping);
```

Файл microservice-constructor/src/writer.ts

```
import { File } from './types/file';
import * as fs from 'fs/promises';
import { existsSync } from 'fs';
import { removeExtraWhiteSpace } from './utils/remove-extra-white-space';

export class FileWriter {
  constructor(private target: string) {}

  async writeFile(file: File): Promise<void> {
    file = { ...file, data: removeExtraWhiteSpace(file.data) };
    //const path = __dirname + '/' + this.target + '/' + file.path;
    const path = this.target + '/' + file.path;
    if (!existsSync(path)) {
      await fs.mkdir(path, { recursive: true });
    }
    await fs.writeFile(path + '/' + file.name, file.data);
  }

  async writeFiles(files: File[]): Promise<void> {
    for (const file of files) {
      await this.writeFile(file);
    }
  }
}
```

Файл microservice-constructor/src/cicd-generator/cicd-generator-factory.ts

```
import { ProjectConfig } from '../types/config/project-config';
import GithubDoCICDGenerator from './github-do-cicd-generator';
import { SetSecretsCommandsProvider } from '../types/set-secrets-commands-
provider';
import { SecretsCreator } from '../secret-creator';
import { GeneratorMapping } from '../types/generator-mapping';
import { GithubDoInfoProvider } from './github-do-cicd-generator/github-do-info-
provider';
import { GeneratorFactory } from '../generator-factory';
import { InfoProviderFactory } from '../info-provider-factory';

const cicdGeneratorMapping: GeneratorMapping<ProjectConfig,
SetSecretsCommandsProvider> = {
  'github-actions-digitalocean': {
    getGenerator(secretsCreator: SecretsCreator, projectConfig: ProjectConfig) {
```

```
      return new GithubDoCICDGenerator(secretsCreator, projectConfig);
    },
    getInfoProvider(projectConfig: ProjectConfig) {
      return new GithubDoInfoProvider(projectConfig);
    },
  },
};

export const cicdGeneratorFactory = new GeneratorFactory(cicdGeneratorMapping);
export const cicdInfoProviderFactory = new
InfoProviderFactory(cicdGeneratorMapping as GeneratorMapping<unknown,
SetSecretsCommandsProvider>);
export const availableCICDSetups = Object.keys(cicdGeneratorMapping);
```

Файл microservice-constructor/src/cicd-generator/github-do-cicd-generator/templates/deploy.yaml.template.ts

```
import { File } from '../../../types/file';
import { FileTemplate } from '../../../types/file-template';
import { ProjectConfig } from '../../../types/config/project-config';
import { ServiceConfig } from '../../../types/config/service-config';

export class DeployYamlTemplate implements FileTemplate<ServiceConfig> {
  constructor(private projectConfig: ProjectConfig) {}

  getFile(serviceConfig: ServiceConfig): File{
    return {
      name: `deploy-${serviceConfig.name}.yaml`,
      path: '.github/workflows',
      data: `
        name: ${serviceConfig.name}

        on:
          push:
            branches: [ main ]
            paths: [ '${serviceConfig.name}/**' ]

        jobs:
          build:
            runs-on: ubuntu-latest
            steps:
            - uses: actions/checkout@v2
            - run: cd ${serviceConfig.name}
            - run: docker login -u $DOCKER_USERNAME -p $DOCKER_PASSWORD
              env:
                DOCKER_USERNAME: \${{ secrets.DOCKER_USERNAME }}
                DOCKER_PASSWORD: \${{ secrets.DOCKER_PASSWORD }}
            - run: docker push
${this.projectConfig.dockerUsername}/${serviceConfig.name}
```

```
            - uses: digitalocean/action-doctl@v2
              with:
                token: \${{ secrets.DIGITALOCEAN_ACCESS_TOKEN }}
            - run: doctl kubernetes cluster kubeconfig save
${this.projectConfig.name}
            - run: kubectl rollout restart deployment ${serviceConfig.name}-depl
      `,
    };
  }
}
```

Файл microservice-constructor/src/cicd-generator/github-do-cicd-generator/templates/tests.yaml.template.ts

```
import { File } from '../../../types/file';
import { FileTemplate } from '../../../types/file-template';
import { ServiceConfig } from '../../../types/config/service-config';
import { addIndentation } from '../../../utils/add-indentation';

export class TestsYamlTemplate implements FileTemplate<ServiceConfig> {
  constructor(private testCommands: string[]) {}

  getFile(serviceConfig: ServiceConfig): File{
    return {
      name: `test-${serviceConfig.name}.yaml`,
      path: '.github/workflows',
      data: `
        name: ${serviceConfig.name}

        on:
          push:
            branches: [ main ]
            paths: [ '${serviceConfig.name}/**' ]
          pull_request:
            branches: [ main ]
            paths: [ '${serviceConfig.name}/**' ]

        jobs:
          test:
            runs-on: ubuntu-latest
            steps:
            - uses: actions/checkout@v2
            - run: cd ${serviceConfig.name}
            ${addIndentation(this.testCommands.map(command => `- run:
${command}`).join('\n'), '\t\t\t\t\t\t', true)}
    `,
    };
  }
}
```

Файл microservice-constructor/src/cicd-generator/github-do-cicd-generator/templates/deploy-manifests.yaml.template.ts

```
import { File } from '../../../types/file';
import { FileTemplate } from '../../../types/file-template';
import { ProjectConfig } from '../../../types/config/project-config';

export class DeployManifestsYamlTemplate implements FileTemplate<ProjectConfig> {
  getFile(projectConfig: ProjectConfig): File{
    return {
      name: 'deploy-manifests.yaml',
      path: '.github/workflows',
      data: `
        name: deploy-manifests

        on:
          push:
            branches: [ main ]
            paths: [ 'infra/**' ]

        jobs:
          build:
            runs-on: ubuntu-latest
            steps:
            - uses: actions/checkout@v2
            - uses: digitalocean/action-doctl@v2
              with:
                token: \${{ secrets.DIGITALOCEAN_ACCESS_TOKEN }}
            - run: doctl kubernetes cluster kubeconfig save ${projectConfig.name}
            - run: kubectl apply -f infra/k8s
      `,
    };
  }
}
```

Файл microservice-constructor/src/cicd-generator/github-do-cicd-generator/index.ts

```
import { serverInfoProviderFactory } from '../../server-generator/server-
generator-factory';
import { SecretsCreator } from '../../secret-creator';
import { File } from '../../types/file';
import { FilesGenerator } from '../../types/files-generator';
import { ProjectConfig } from '../../types/config/project-config';
import { ServiceConfig } from '../../types/config/service-config';
import { DeployManifestsYamlTemplate } from './templates/deploy-
manifests.yaml.template';
```

```
import { DeployYamlTemplate } from './templates/deploy.yaml.template';
import { TestsYamlTemplate } from './templates/tests.yaml.template';

export default class GithubDoCICDGenerator implements
FilesGenerator<ProjectConfig>  {
  constructor(
    private secretsCreator: SecretsCreator,
    private projectConfig: ProjectConfig,
  ) {}

  generateFiles(config: ProjectConfig): File[] {
    this.addSecrets();
    const files: File[] = [];
    const deployTemplate = new DeployYamlTemplate(config);
    const deployManifestsTemplate = new DeployManifestsYamlTemplate();
    for (const service of config.services) {
      files.push(this.generateTestWorkflow(service));
      files.push(deployTemplate.getFile(service));
    }
    files.push(deployManifestsTemplate.getFile(config));
    return files;
  }

  generateTestWorkflow(serviceConfig: ServiceConfig): File {
    const serverInfoProvider = serverInfoProviderFactory.get(serviceConfig.lang,
this.projectConfig);
    const testCommands = serverInfoProvider.getTestCommands();
    const serviceTestWorkflow = new
TestsYamlTemplate(testCommands).getFile(serviceConfig);
    return serviceTestWorkflow;
  }

  addSecrets(): void {
    this.secretsCreator.addSecret('GH_TOKEN');
  }
}
```

Файл microservice-constructor/src/cicd-generator/github-do-cicd-generator/github-do-info-provider.ts

```
import { ProjectConfig } from '../../types/config/project-config';
import { Secret } from '../../types/secret';
import { SetSecretsCommandsProvider } from '../../types/set-secrets-commands-
provider';

export class GithubDoInfoProvider implements SetSecretsCommandsProvider {
  constructor(private projectConfig: ProjectConfig) {}

  getSetSecretsCommands(secrets: Secret[]): string[] {
```

```
    const commands = ['gh auth login --with-token $GH_TOKEN'];
    commands.push(...secrets.map(secret => `gh secret set ${secret.name} --body
"$${secret.name}"`));
    return commands;
  }
}
```

Файл microservice-constructor/src/generator-factory.ts

```
import { SecretsCreator } from './secret-creator';
import { FilesGenerator } from './types/files-generator';
import { GeneratorMapping } from './types/generator-mapping';
import { ProjectConfig } from './types/config/project-config';

export class GeneratorFactory<T> {
  constructor(public generatorMapping: GeneratorMapping<T, unknown>) {}

  get(type: string, secretsCreator: SecretsCreator, projectConfig:
ProjectConfig): FilesGenerator<T> {
    const getGenerator = this.generatorMapping[type].getGenerator;
    if (!getGenerator) {
      throw new Error(`${type} is not supported`);
    }
    return getGenerator(secretsCreator, projectConfig);
  }
}
```

Файл microservice-constructor/src/server-generator/ts-nest-server-generator/get-type-name.ts

```
import { Capitalize } from '../../utils/case-utils';
import { EndpointGroup, isEntityRef, Value } from '../../types/config/api-
config';

const typeMap: { [key: string]: string } = {
  'string': 'string',
  'number': 'number',
};

export function getTypeName(value: Value, endpointGroup: EndpointGroup, imports:
string[]): string {
  const { type } = value;
  if (isEntityRef(type)) {
    const entityName = endpointGroup.entities.find(entity => entity.id ===
type.id)?.name;
    if (!entityName) {
      throw new Error('Invalid entity reference');
    }
```

```
    imports.push(Capitalize(entityName));
    return Capitalize(entityName);
  }
  if (type !== 'array') {
    return typeMap[type];
  }
  const itemType = value.items;
  if (itemType === undefined) {
    throw new Error('Type of value is array, but type of item is not specified');
  }
  return getTypeName(itemType, endpointGroup, imports) + '[]';
}
```

Файл microservice-constructor/src/server-generator/ts-nest-server-generator/templates/module/controller.template.ts

```
import { removeExtraWhiteSpace } from '../../../../utils/remove-extra-white-
space';
import { File } from '../../../../types/file';
import { FileTemplate } from '../../../../types/file-template';
import { Endpoint, EndpointGroup, Value } from '../../../../types/config/api-
config';
import { ServiceConfig } from '../../../../types/config/service-config';
import { getTypeName } from '../../get-type-name';
import { getParams } from '../../path-parsing';
import { addIndentation } from '../../../../utils/add-indentation';
import { Capitalize, Decapitalize } from '../../../../utils/case-utils';
import { getDistinct } from '../../../../utils/distinct';
import { removeEmptyLines } from '../../../../utils/remove-empty-lines';
import { isArray } from '../../../../utils/is-array';
import { getNonArrayType } from '../../../../utils/get-array-item-type';

export class ControllerTemplate implements FileTemplate<EndpointGroup> {
  constructor(
    private serviceConfig: ServiceConfig,
  ) { }

  getFile(endpointGroup: EndpointGroup): File {
    return {
      name: `${endpointGroup.name}.controller.ts`,
      path: `${this.serviceConfig.name}/src/${endpointGroup.name}/`,
      data: this.generateEndpoints(endpointGroup),
    };
  }

  private generateEndpoints(endpointGroup: EndpointGroup): string {
    const imports: string[] = [];
```

```
    const endpoints = endpointGroup.endpoints.map(endpoint =>
this.generateEndpoint(endpoint, endpointGroup, imports));

    const dto = removeExtraWhiteSpace(`
      ${this.serviceConfig.docs ? `@ApiTags('${endpointGroup.name}')` : ''}
      @Controller('${endpointGroup.prefix}')
      export class ${Capitalize(endpointGroup.name)}Controller {
        ${addIndentation(endpoints.join('\n\n'), '\t\t\t\t', true)}
      }`);

    const nestJsImports = this.generateNestjsImports(endpointGroup);

    const swaggerImports = this.serviceConfig.docs ? "import { ApiTags,
ApiResponse } from '@nestjs/swagger';\n" : '';

    return nestJsImports + '\n' + swaggerImports + getDistinct(imports).map(name
=> `import { ${name} } from './dto/${Decapitalize(name)}.dto';`).join('\n') +
'\n' + dto;
  }

  private generateEndpoint(endpoint: Endpoint, endpointGroup: EndpointGroup,
imports: string[]): string {
    const bodyType = this.getExchangeContentType(endpoint.request?.content,
endpointGroup, imports);
    const responseType = this.getExchangeContentType(endpoint.response?.content,
endpointGroup, imports);
    const body = bodyType ? `@Body() body: ${bodyType}` : undefined;
    const params = getParams(endpoint).map(param => `@Param('${param}') ${param}:
string`);
    if (body) params.unshift(body);
    const statusCode = endpoint.response?.statusCode;
    const statusCodeStr = `@HttpCode(${statusCode})`;

    return removeEmptyLines(removeExtraWhiteSpace(`
      @${Capitalize(endpoint.method)}('${endpoint.path}')
      ${statusCode ? '\n\t\t\t' + statusCodeStr : ''}
      ${this.serviceConfig.docs && (statusCode || responseType) ? `
      @ApiResponse({
        ${statusCode ? `status: ${statusCode},` : ''}
        ${responseType ? `type: ${getNonArrayType(responseType)},` : ''}
        ${isArray(responseType || '') ? 'isArray: true,' : ''}
      })
      ` : ''}
      ${endpoint.name}(${params.join(', ')}): ${responseType || 'void'} {
        // TODO: add business logic
      }`));
  }

  private getExchangeContentType(value: Value | undefined, endpointGroup:
EndpointGroup, imports: string[]): string | undefined {
```

```
    if (value === undefined) {
      return undefined;
    }
    return getTypeName(value, endpointGroup, imports);
  }

  private generateNestjsImports(endpointGroup: EndpointGroup): string {
    const imports = ['Controller'];
    for (const endpoint of endpointGroup.endpoints) {
      imports.push(endpoint.method);
      if (endpoint.path.includes(':')) imports.push('Param');
      if (endpoint.request) imports.push('Body');
      if (endpoint.response?.statusCode) imports.push('HttpCode');
    }
    return `import { ${getDistinct(imports).join(', ')} } from
'@nestjs/common';`;
  }
}
```

Файл microservice-constructor/src/server-generator/ts-nest-server-generator/templates/module/module.template.ts

```
import { Capitalize } from '../../../../utils/case-utils';
import { File } from '../../../../types/file';
import { FileTemplate } from '../../../../types/file-template';
import { EndpointGroup } from '../../../../types/config/api-config';
import { ServiceConfig } from '../../../../types/config/service-config';

export class ModuleTemplate implements FileTemplate<EndpointGroup> {
  constructor(
    private serviceConfig: ServiceConfig,
  ) { }

  getFile(endpointGroup: EndpointGroup): File {
    return {
      name: `${endpointGroup.name}.module.ts`,
      path: `${this.serviceConfig.name}/src/${endpointGroup.name}/`,
      data: `
      import { Module } from '@nestjs/common';
      import { ${Capitalize(endpointGroup.name)}Controller } from
'./${endpointGroup.name}.controller';

      @Module({
        controllers: [${Capitalize(endpointGroup.name)}Controller],
      })
      export class ${Capitalize(endpointGroup.name)}Module {}
      `,
    };
  }
```

```
}
```

Файл microservice-constructor/src/server-generator/ts-nest-server-generator/templates/module/dto.template.ts

```
import { removeExtraWhiteSpace } from '../../../../utils/remove-extra-white-
space';
import { File } from '../../../../types/file';
import { FileTemplate } from '../../../../types/file-template';
import { EndpointGroup, Entity, Field, Value } from
'../../../../types/config/api-config';
import { ServiceConfig } from '../../../../types/config/service-config';
import { getTypeName } from '../../get-type-name';
import { addIndentation } from '../../../../utils/add-indentation';
import { Capitalize, Decapitalize } from '../../../../utils/case-utils';
import { getDistinct } from '../../../../utils/distinct';
import { removeEmptyLines } from '../../../../utils/remove-empty-lines';
import { isArray } from '../../../../utils/is-array';
import { getNonArrayType } from '../../../../utils/get-array-item-type';

export class DtoTemplate implements FileTemplate<Entity> {
  constructor(
    private endpointGroup: EndpointGroup,
    private serviceConfig: ServiceConfig,
  ) { }

  getFile(entity: Entity): File {
    return {
      name: `${entity.name}.dto.ts`,
      path: `${this.serviceConfig.name}/src/${this.endpointGroup.name}/dto`,
      data: this.generateDto(entity),
    };
  }

  private generateDto(entity: Entity): string {
    const imports: string[] = [];

    const fields = entity.fields.map(field => this.generateField(field,
imports));

    const dto = removeExtraWhiteSpace(`
      ${this.serviceConfig.docs ? "import { ApiProperty } from
'@nestjs/swagger';" : ''}

      export class ${Capitalize(entity.name)} {
        ${addIndentation(fields.join('\n\n'), '\t\t\t\t', true)}
      }`);
```

```
    return getDistinct(imports).map(name => `import { ${name} } from
'./${Decapitalize(name)}.dto';`).join('\n') + dto;
  }

  private generateField(field: Field, imports: string[]): string {
    const typeName = getTypeName(field as Value, this.endpointGroup, imports);
    return removeEmptyLines(removeExtraWhiteSpace(`
      ${this.serviceConfig.docs ? `@ApiProperty(${isArray(typeName) ? `{ type:
${getNonArrayType(typeName)}, isArray: true }` : ''})` : ''}
      ${field.name}: ${typeName};
    `));
  }
}
```

Файл microservice-constructor/src/server-generator/ts-nest-server-generator/templates/service/app.module.template.ts

```
import { addIndentation } from '../../../../utils/add-indentation';
import { Capitalize } from '../../../../utils/case-utils';
import { File } from '../../../../types/file';
import { FileTemplate } from '../../../../types/file-template';
import { ServiceConfig } from '../../../../types/config/service-config';

export class AppModuleTsTemplate implements FileTemplate<ServiceConfig> {
  getFile(config: ServiceConfig): File {
    return {
      name: 'app.module.ts',
      path: `${config.name}/src`,
      data: `
        import { Module } from '@nestjs/common';
        import { AppController } from './app.controller';
        ${this.generateModuleImports(config)}

        @Module({
          imports: [${config.api ? config.api.endpointGroups.map(({ name }) =>
`${Capitalize(name)}Module`) : ''}],
          controllers: [AppController],
          providers: [],
        })
        export class AppModule {}
      `,
    };
  }

  generateModuleImports(config: ServiceConfig): string {
    if (!config.api) return '';

    const imports = config.api.endpointGroups
```

```
      .map(({ name }) => `import { ${Capitalize(name)}Module } from
'./${name}/${name}.module';` )
      .join('\n');

    return addIndentation(imports, '\t\t\t\t', true);
  }
}
```

Файл microservice-constructor/src/server-generator/ts-nest-server-generator/templates/service/docker-file.template.ts

```
import { File } from '../../../../types/file';
import { FileTemplate } from '../../../../types/file-template';
import { ServiceConfig } from '../../../../types/config/service-config';

export class DockerfileTemplate implements FileTemplate<ServiceConfig> {
  getFile(config: ServiceConfig): File {
    return {
      name: 'Dockerfile',
      path: `${config.name}`,
      data: `
        FROM node:alpine

        WORKDIR /app
        COPY package.json .
        RUN npm install --only=prod
        COPY . .

        CMD ["npm", "start"]
      `,
    };
  }
}
```

Файл microservice-constructor/src/server-generator/ts-nest-server-generator/path-parsing.ts

```
import { Endpoint } from '../../types/config/api-config';

export function getParams(endpoint: Endpoint): string[] {
  return endpoint.path
    .split('/')
    .filter(str => str[0] === ':')
    .map(str => str.slice(1));
}
```

Файл microservice-constructor/src/server-generator/ts-nest-server-generator/index.ts

```
import { SecretsCreator } from '../../secret-creator';
import { ProjectConfig } from '../../types/config/project-config';
import { File } from '../../types/file';
import { FileTemplate } from '../../types/file-template';
import { FilesGenerator } from '../../types/files-generator';
import { ServiceConfig } from '../../types/config/service-config';
import ModuleGenerator from './module-generator';
import { AppControllerTsTemplate } from
'./templates/service/app.controller.template';
import { AppModuleTsTemplate } from './templates/service/app.module.template';
import { DockerfileTemplate } from './templates/service/docker-file.template';
import { DockerignoreTemplate } from './templates/service/dockerignore.template';
import { EslintrcTemplate } from './templates/service/eslintrc.template';
import { MainTsTemplate } from './templates/service/main.template';
import { PackageJsonTemplate } from './templates/service/package.json.template';
import { PrettierrcTemplate } from './templates/service/prettierrc.template';
import { TsconfigBuildJsonTemplate } from
'./templates/service/tsconfig.build.template';
import { TsconfigJsonTemplate } from './templates/service/tsconfig.template';

export default class TsNestServerGenerator implements
FilesGenerator<ServiceConfig> {
  constructor(
    private secretsCreator: SecretsCreator,
    private projectConfig: ProjectConfig,
  ) {}

  templates: FileTemplate<ServiceConfig>[] = [
    new AppControllerTsTemplate(),
    new AppModuleTsTemplate(),
    new DockerfileTemplate(),
    new DockerignoreTemplate(),
    new EslintrcTemplate(),
    new MainTsTemplate(),
    new PackageJsonTemplate(),
    new PrettierrcTemplate(),
    new TsconfigJsonTemplate(),
    new TsconfigBuildJsonTemplate(),
  ];

  generateFiles(config: ServiceConfig): File[] {
    const files: File[] = [];
    for (const template of this.templates) {
      files.push(template.getFile(config));
    }
    const moduleGenerator = new ModuleGenerator(config);
```

```
    const apiConfig = config.api;
    if (apiConfig) {
      for (const endpointGroup of apiConfig.endpointGroups) {
        files.push(...moduleGenerator.generateFiles(endpointGroup));
      }
    }
    return files;
  }
}
```

Файл microservice-constructor/src/server-generator/ts-nest-server-generator/module-generator.ts

```
import { File } from '../../types/file';
import { FileTemplate } from '../../types/file-template';
import { FilesGenerator } from '../../types/files-generator';
import { EndpointGroup } from '../../types/config/api-config';
import { ServiceConfig } from '../../types/config/service-config';
import { ControllerTemplate } from './templates/module/controller.template';
import { DtoTemplate } from './templates/module/dto.template';
import { ModuleTemplate } from './templates/module/module.template';

export default class ModuleGenerator implements FilesGenerator<EndpointGroup> {
  constructor(private serviceConfig: ServiceConfig) {}

  templates: FileTemplate<EndpointGroup>[] = [
    new ControllerTemplate(this.serviceConfig),
    new ModuleTemplate(this.serviceConfig),
  ];

  generateFiles(endpointGroup: EndpointGroup): File[] {
    const files: File[] = [];
    for (const template of this.templates) {
      files.push(template.getFile(endpointGroup));
    }
    const dtoTemplate = new DtoTemplate(endpointGroup, this.serviceConfig);
    for (const entity of endpointGroup.entities) {
      files.push(dtoTemplate.getFile(entity));
    }
    return files;
  }
}
```

Файл microservice-constructor/src/server-generator/ts-nest-server-generator/ts-nest-info-provider.ts

```
import { ProjectConfig } from '../../types/config/project-config';
import { TestCommandsProvider } from '../../types/test-command-provider';
```

```
export class TsNestInfoProvider implements TestCommandsProvider {
  constructor(private projectConfig: ProjectConfig) {}

  getTestCommands(): string[] {
    return [
      'npm install',
      'npm run test',
    ];
  }
}
```

Файл microservice-constructor/src/server-generator/server-generator-factory.ts

```
import { TestCommandsProvider } from '../types/test-command-provider';
import { ServiceConfig } from '../types/config/service-config';
import TsNestServerGenerator from './ts-nest-server-generator';
import { GeneratorMapping } from '../types/generator-mapping';
import { TsNestInfoProvider } from './ts-nest-server-generator/ts-nest-info-
provider';
import { SecretsCreator } from '../secret-creator';
import { ProjectConfig } from '../types/config/project-config';
import { GeneratorFactory } from '../generator-factory';
import { InfoProviderFactory } from '../info-provider-factory';

export const serverGeneratorMapping: GeneratorMapping<ServiceConfig,
TestCommandsProvider> = {
  'ts-nest': {
    getGenerator(secretsCreator: SecretsCreator, projectConfig: ProjectConfig) {
      return new TsNestServerGenerator(secretsCreator, projectConfig);
    },
    getInfoProvider(projectConfig: ProjectConfig) {
      return new TsNestInfoProvider(projectConfig);
    },
  },
};

export const serverGeneratorFactory = new
GeneratorFactory(serverGeneratorMapping);
export const serverInfoProviderFactory = new
InfoProviderFactory(serverGeneratorMapping as GeneratorMapping<unknown,
TestCommandsProvider>);
export const availableServiceSetups = Object.keys(serverGeneratorMapping);
```

Файл microservice-constructor/src/secret-creator/templates/secrets.sh.template.ts

```
import { Secret } from '../../types/secret';
import { File } from '../../types/file';
import { FileTemplate } from '../../types/file-template';

export class SecretsShTemplate implements FileTemplate<Secret[]> {
  getFile(secrets: Secret[]): File {
    return {
      name: 'secrets.sh',
      path: '/',
      data: secrets.map(secret => `${secret.name}=${secret.value}`).join('\n'),
    };
  }
}
```

Файл microservice-constructor/src/secret-creator/index.ts

```
import prompt from 'prompt';
import { cicdInfoProviderFactory } from '../cicd-generator/cicd-generator-
factory';
import { File } from '../types/file';
import { FilesGenerator } from '../types/files-generator';
import { ProjectConfig } from '../types/config/project-config';
import { Secret } from '../types/secret';
import { SecretsShTemplate } from './templates/secrets.sh.template';

export class SecretsCreator implements FilesGenerator<ProjectConfig> {
  secrets: Secret[] = [];

  addSecret(secret: string): void {
    this.secrets.push({ name: secret, value: '' });
  }

  async promptSecrets(): Promise<void> {
    prompt.start();

    for (const secret of this.secrets) {
      secret.value = await this.promptSecret(secret.name);
    }
  }

  async promptSecret(secretName: string): Promise<string> {
    const str = (await prompt.get({ name: secretName, message: `Please enter
${secretName}` }))[secretName];
    if (typeof str === 'string') {
      return str;
    }
    throw new Error('');
  }

  generateFiles(config: ProjectConfig): File[] {
```

```
    const cicdGenerator = cicdInfoProviderFactory.get(config.cicd, config);
    const secretCommands = cicdGenerator.getSetSecretsCommands(this.secrets);
    const file = new SecretsShTemplate().getFile(this.secrets);
    file.data += '\n\n\n' + secretCommands.join('\n');
    return [file];
  }
}
```

Файл microservice-constructor/src/project-generator.ts

```
import { cicdGeneratorFactory } from './cicd-generator/cicd-generator-factory';
import { databaseGeneratorFactory } from './database-generator/database-
generator-factory';
import { eventBusGeneratorFactory } from './event-bus-generator/event-bus-
generator-factory';
import { GatewayGenerator } from './gateway-generator';
import { SecretsCreator } from './secret-creator';
import { serverGeneratorFactory } from './server-generator/server-generator-
factory';
import { ServiceK8SConfigGenerator } from './service-k8s-config-
generator/service-k8s-config-generator';
import { File } from './types/file';
import { ProjectConfig } from './types/config/project-config';
import { getFilesGetter } from './utils/get-files-getter';


export async function generateProject(projectConfig: ProjectConfig):
Promise<File[]> {
  const secretsCreator = new SecretsCreator();

  const getFiles = getFilesGetter(secretsCreator, projectConfig);
  const serviceK8SConfigGenerator = new ServiceK8SConfigGenerator(projectConfig);

  const files = [
    ...getFiles(serverGeneratorFactory, projectConfig.services, 'lang'),
    ...getFiles(databaseGeneratorFactory, projectConfig.databases, 'type'),
    ...getFiles(eventBusGeneratorFactory, projectConfig.eventBuses, 'type'),
    ...getFiles(cicdGeneratorFactory, projectConfig, 'cicd'),
    ...new GatewayGenerator().generateFiles(projectConfig),
    ...projectConfig.services.flatMap(service =>
serviceK8SConfigGenerator.generateFiles(service)),
  ];

  await secretsCreator.promptSecrets();
  files.push(...secretsCreator.generateFiles(projectConfig));

  return files;
}
```

Файл microservice-constructor/src/database-generator/postgres-database-generator/templates/postgres-vol.yaml.template.ts

```
import { Database } from '../../../types/config/database';
import { File } from '../../../types/file';
import { FileTemplate } from '../../../types/file-template';

export class PostgresVolTemplate implements FileTemplate<Database> {
  getFile(database: Database): File {
    return {
      name: `${database.name}-postgres-vol.yaml`,
      path: 'infra/',
      data: `
        kind: PersistentVolume
        apiVersion: v1
        metadata:
          name: ${database.name}-postgres-volume
          labels:
            type: local
            app: ${database.name}-postgres
        spec:
          storageClassName: manual
          capacity:
            storage: ${database.sizeGb}Gi
          accessModes:
            - ReadWriteOnce
          hostPath:
            path: "/mnt/data"
        ---
        apiVersion: v1
        kind: "PersistentVolumeClaim"
        metadata:
          name: "${database.name}-postgres-data-claim"
        spec:
          accessModes:
          - ReadWriteOnce
          resources:
            requests:
              storage: ${database.sizeGb}Gi
              storageClassName: manual
      `,
    };
  }
}
```

Файл microservice-constructor/src/database-generator/postgres-database-generator/templates/postgres-password.yaml.template.ts

```
import { Database } from '../../../types/config/database';
import { File } from '../../../types/file';
import { FileTemplate } from '../../../types/file-template';

export class PostgresPasswordTemplate implements FileTemplate<Database> {
  getFile(database: Database): File {
    return {
      name: `${database.name}-postgres-password.yaml`,
      path: 'infra/',
      data: `
        apiVersion: v1
        kind: Secret
        metadata:
          name: ${database.name}-postgres-password-secret
        type: Opaque
        data:
        ${database.name.toUpperCase()}_POSTGRES_PASSWORD:
$${database.name.toUpperCase()}_POSTGRES_PASSWORD_BASE64
      `,
    };
  }
}
```

Файл microservice-constructor/src/database-generator/postgres-database-generator/templates/postgres-depl.yaml.template.ts

```
import { Database } from '../../../types/config/database';
import { File } from '../../../types/file';
import { FileTemplate } from '../../../types/file-template';

export class PostgresDeplTemplate implements FileTemplate<Database> {
  getFile(database: Database): File {
    return {
      name: `${database.name}-postgres-depl.yaml`,
      path: 'infra/',
      data: `
        apiVersion: v1
        kind: Deployment
        metadata:
          name: ${database.name}-postgres-depl
          labels:
            app: ${database.name}-postgres-depl
        spec:
          replicas: 1
          template:
            metadata:
              labels:
                app: ${database.name}-postgres
            spec:
```

```
              containers:
              - image: postgres:9.4
                name: ${database.name}-postgres
                ports:
                - containerPort: 5432
                volumeMounts:
                - name: ${database.name}-postgres-data
                  mountPath: /var/lib/postgresql
                env:
                  - name: POSTGRES_DB
                    value: '${database.name}'
                  - name: POSTGRES_USER
                    value: '${database.name}-admin'
                  - name: POSTGRES_PASSWORD
                    valueFrom:
                      secretKeyRef:
                        name: ${database.name}-password-secret
                        key:
${database.name.toUpperCase()}_POSTGRES_PASSWORD
              volumes:
              - name: ${database.name}-postgres-data
                persistentVolumeClaim:
                  claimName: ${database.name}-postgres-data-claim
        ---
        apiVersion: v1
        kind: Service
        metadata:
          name: ${database.name}-postgres-serv
          labels:
            name: ${database.name}-postgres-serv
        spec:
          type : ClusterIP
          ports:
            - name: ${database.name}-postgres
              port: 5432
              targetPort : 5432
          selector:
            app: ${database.name}-postgres
      `,
    };
  }
}
```

Файл microservice-constructor/src/database-generator/postgres-database-generator/index.ts

```
import { ProjectConfig } from '../../types/config/project-config';
import { SecretsCreator } from '../../secret-creator';
import { Database } from '../../types/config/database';
```

```
import { File } from '../../types/file';
import { FileTemplate } from '../../types/file-template';
import { FilesGenerator } from '../../types/files-generator';
import { PostgresDeplTemplate } from './templates/postgres-depl.yaml.template';
import { PostgresPasswordTemplate } from './templates/postgres-
password.yaml.template';
import { PostgresVolTemplate } from './templates/postgres-vol.yaml.template';

export class PostgresDatabaseGenerator implements FilesGenerator<Database> {
  constructor(
    private secretsCreator: SecretsCreator,
    private projectConfig: ProjectConfig,
  ) {}

  templates: FileTemplate<Database>[] = [
    new PostgresDeplTemplate(),
    new PostgresVolTemplate(),
    new PostgresPasswordTemplate(),
  ];

  generateFiles(db: Database): File[] {
    this.addSecrets(db);
    const files = this.templates.map(template => template.getFile(db));
    return files;
  }

  addSecrets(db: Database): void {
    this.secretsCreator.addSecret(`${db.name.toUpperCase()}_POSTGRES_PASSWORD`);
    this.secretsCreator.addSecret(`${db.name.toUpperCase()}_POSTGRES_PASSWORD_BAS
E64`);
  }
}
```

Файл microservice-constructor/src/database-generator/postgres-database-generator/postgres-info-provider.ts

```
import { EnvVariableProvider, SecretProvider } from '../../types/env-provider';
import { EnvVariable } from '../../types/env-variable';
import { ProjectConfig } from '../../types/config/project-config';

export class PostgresInfoProvider implements EnvVariableProvider, SecretProvider
{
  constructor(private projectConfig: ProjectConfig) {}

  getSecrets(id: number): string[] {
    const db = this.projectConfig.databases.find(database => database.id === id);
    if (!db) throw new Error(`Invalid reference, database with id = ${id} is not
found`);
    return [`${db.name.toUpperCase()}_POSTGRES_PASSWORD`];
```

```
  }

  getEnvVariables(id: number): EnvVariable[] {
    const db = this.projectConfig.databases.find(database => database.id === id);
    if (!db) throw new Error(`Invalid reference, database with id = ${id} is not
found`);
    return [
      { name: 'POSTGRES_DB', value: db.name },
      { name: 'POSTGRES_USER', value: `${db.name}-admin` },
    ];
  }
}
```

Файл microservice-constructor/src/database-generator/database-generator-factory.ts

```
import { Database } from '../types/config/database';
import { PostgresDatabaseGenerator } from './postgres-database-generator';
import { SecretsCreator } from '../secret-creator';
import { EnvVariableProvider, SecretProvider } from 'src/types/env-provider';
import { GeneratorMapping } from 'src/types/generator-mapping';
import { PostgresInfoProvider } from './postgres-database-generator/postgres-
info-provider';
import { ProjectConfig } from '../types/config/project-config';
import { GeneratorFactory } from '../generator-factory';
import { InfoProviderFactory } from '../info-provider-factory';

export const databaseGeneratorMapping: GeneratorMapping<Database,
EnvVariableProvider & SecretProvider> = {
  'postgres': {
    getGenerator(secretsCreator: SecretsCreator, projectConfig: ProjectConfig) {
      return new PostgresDatabaseGenerator(secretsCreator, projectConfig);
    },
    getInfoProvider(projectConfig: ProjectConfig) {
      return new PostgresInfoProvider(projectConfig);
    },
  },
};

export const databaseGeneratorFactory = new
GeneratorFactory(databaseGeneratorMapping);
export const databaseInfoProviderFactory = new
InfoProviderFactory(databaseGeneratorMapping as GeneratorMapping<unknown,
EnvVariableProvider & SecretProvider>);
export const availableDatabases = Object.keys(databaseGeneratorMapping);
```

інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2022 р.

Програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою

Технічне завдання

КПІ.ІП-8321.045490.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Максим ГОЛОВЧЕНКО

Нормоконтроль:

_____ Максим ГОЛОВЧЕНКО

Виконавець:

_____ Владислав ХМАРА

Київ – 2022

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою. Консольний додаток та веб-додаток написано мовою програмування Typescript на платформі NodeJs. Для клієнтської частини веб-додатку використовуються фреймворк Angular, серверної частини – фреймворк NestJs.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка збереження даних;
- перевірка сумісності веб-додатку з останніми версіями сучасних браузерів (Chrome, Opera, Firefox);
- перевірка сумісності консольного додатку з різними операційними системами (Windows, Linux);
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;
- динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;
- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;
- системне тестування – перевіряється уся система в цілому;
- мануальне тестування – тестування без використання автоматизації;
- тестування «білої скриньки» – об’єктом тестування тут є внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується вручну, з метою знаходження помилок та недоліків у функціональній частині програмного забезпечення, в зручності в користуванні та у сумісності з різними браузерами та операційними системами. Для того, щоб перевірити працездатність додатку, необхідно провести наступні тестування:

- динамічне тестування на відповідність функціональним вимогам;
- статичне тестування коду програмного забезпечення;
- тестування веб-додатку на різних браузерах;
- тестування веб-додатку на різних версіях одного браузера;
- тестування консольного додатку на різних операційних системах;
- тестування інтерфейсу користувача;
- тестування зручності використання.

інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2022 р.

Програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою

Технічне завдання

КПІ.ІП-8321.045490.05.34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Максим ГОЛОВЧЕНКО

Нормоконтроль:

_____ Максим ГОЛОВЧЕНКО

Виконавець:

_____ Владислав ХМАРА

Київ – 2022

ЗМІСТ

1 ЗАГАЛЬНІ ВІДОМОСТІ

У склад програмного забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою входять веб-застосунок і консольний застосунок (рисунок 2.3).

Веб-застосунок надає користувачу графічний інтерфейс для конструювання опису бажаного застосунку і можливість отримати конфігураційний файл з цим описом.

Консольний додаток приймає на вхід конфігураційний файл з описом бажаного застосунку і генерує його макет. Макет включає конфігураційні файли для розгортання мікросервісів, баз даних, брокерів повідомлень, шлюзів API, а також файли з частиною коду самих мікросервісів.

2 ПІДГОТОВКА ДО РОБОТИ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

- наявність ПК з операційною системою Windows або Linux;
- наявність останньої версії браузеру Chrome, Firefox або Opera;
- наявність доступу до Інтернету;

2.2 Завантаження застосунку

Отримати доступ до веб-додатку можна перейшовши за адресою https://microservice-constructor-front.herokuapp.com. Консольний застосунок на даний момент застосунок можна встановити використовуючи відповідний deb-пакет, якщо операційна система підтримує їх використання, або ж безпосередньо отримати виконуваний файл для Windows або для дистрибутивів Linux, що не працюють з deb-пакетами. Для цього спершу необхідно завантажити deb-пакет на ПК, а потім, використовуючи apt або dpkg, виконати встановлення пакету.

2.3 Перевірка коректної роботи

По завершенню встановлення пакета треба ввести у консоль команду «microservice-constructor –help». Якщо встановлення відбулось успішно, то у консолі відобразиться довідкова інформація про додаток. У разі відображення помилки встановлення відбулось не успішно. Інакше користувач має змогу взаємодіяти з додатком використовуючи команду «microservice constructor» та вводячи бажані аргументи.

3 РОБОТА ІЗ ЗАСТОСУНКОМ

При вході у веб-застосунок користувачу буде відображена форма авторизації (рисунок 3.1), що містить два поля – поле для введення електронної пошти і поле для паролю. Якщо вони будуть введені коректно, то після натиснення кнопки “Submit” користувач буде перенаправлений на сторінку управління проектами, інакше отримає повідомлення про помилку (рисунок 3.2).

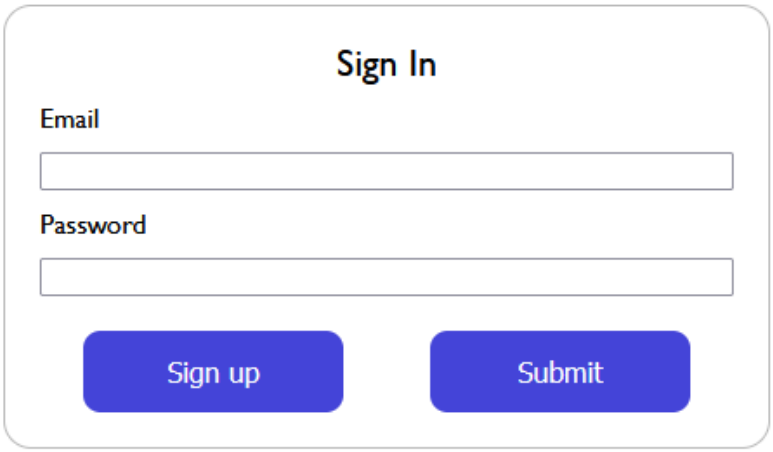


Рисунок 3.1 – Форма авторизації

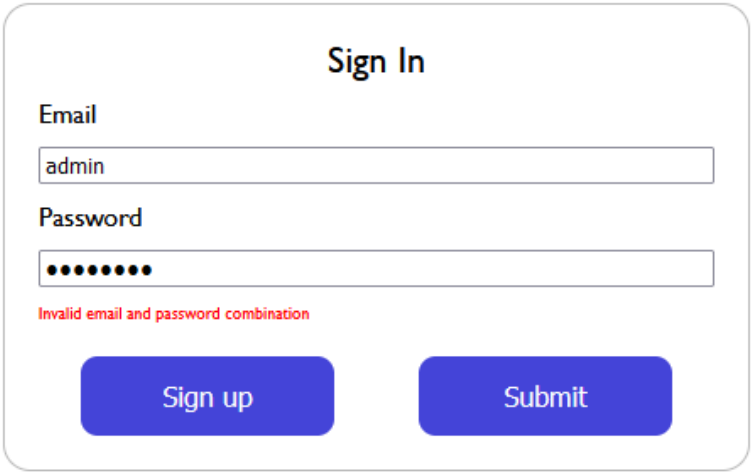


Рисунок 3.2 – Форма авторизації з помилкою

Якщо користувач не зареєстрований, то для створення аккаунту треба натиснути на кнопку “Sign up”. Це перенаправить користувача на сторінку з формою реєстрації (рисунок 3.3). Форма містить три поля: електронна пошта, пароль, підтвердження паролю. Електронна пошта повинна відповідати формату електронної пошти та на неї не повинна використовуватись для

реєстрації повторно. Пароль повинен бути від 10 до 72 символів у довжину, повинен мати хоча б одну англійську літеру, одне число та один спеціальний символ, а також не знаходитись у списку 10000 найпопулярніших паролей. Поле підтвердження паролю повинне відповідати паролю. Якщо всі ці умови виконуються, то після натискання кнопки “Submit” користувач буде перенаправлений на сторінку авторизації, інакше отримає повідомлення про помилку (рисунок 3.4).

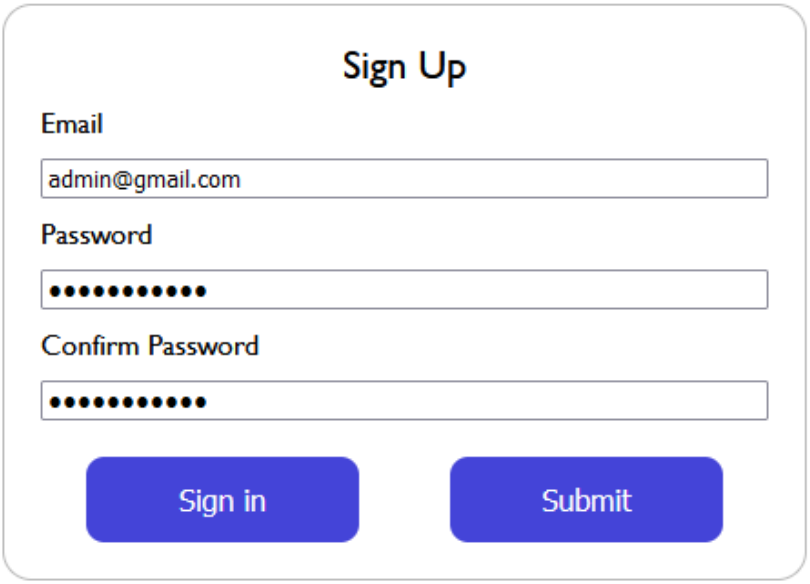


Рисунок 3.3 – Форма реєстрації

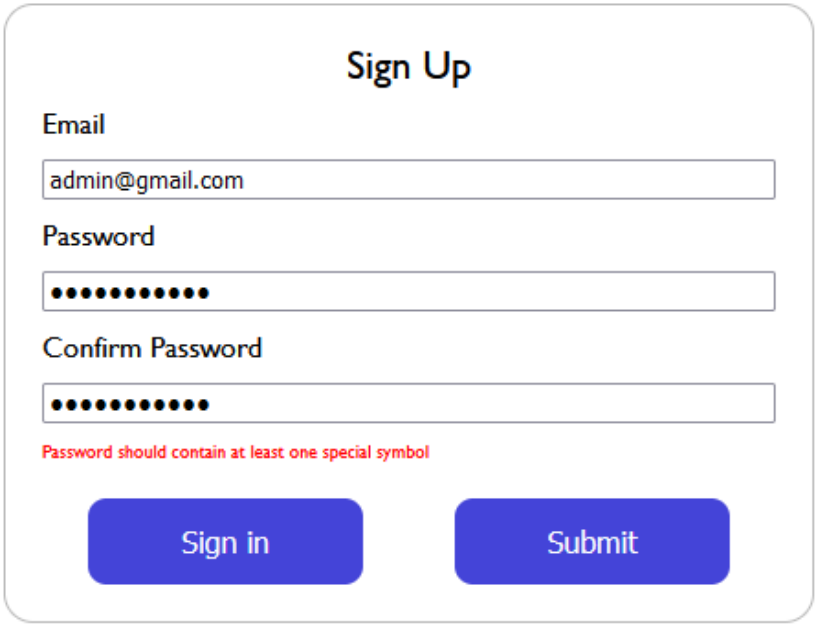


Рисунок 3.4 – Форма реєстрації з повідомленням про помилку

Після потрапляння на сторінку управління (рисунок 3.5) проектами користувач має змогу обрати проект зі списку натиснувши на нього, та перейти на сторінку редагування проекту натиснувши на кнопку “Load”. Також при виборі проекту зі списку можна побачити інформацію про дату його створення та останнього оновлення. Створити новий проект можна за допомогою кнопки “New Project”. Крім того, якщо користувач натисне на кнопку Logout, то він закінчить сесію і зможе обрати інший аккаунт.

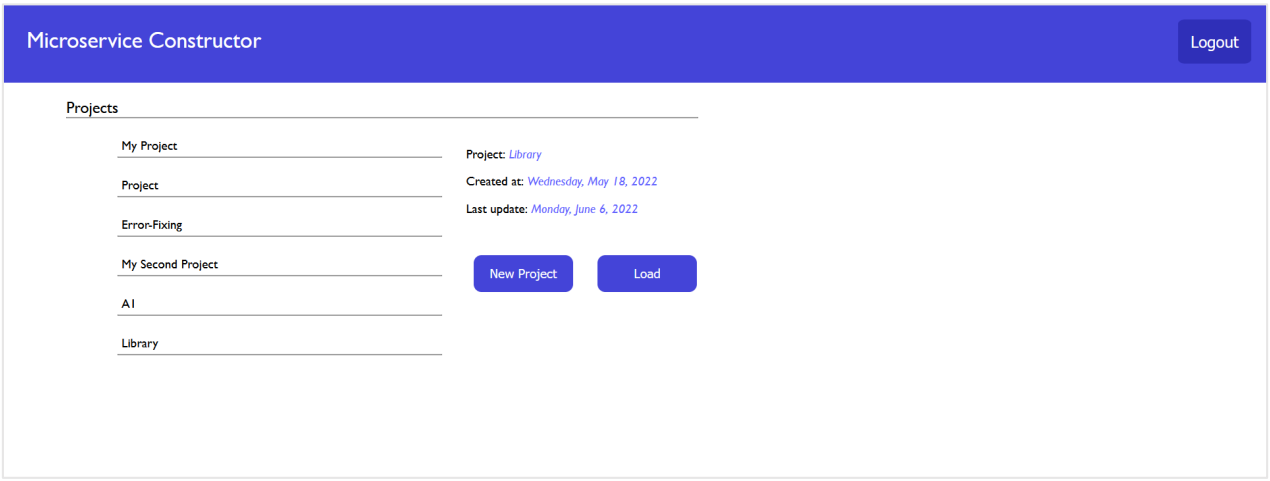


Рисунок 3.5 – Сторінка керування проектами

Всі основні операції відбуваються на сторінці редагування проекту (рисунок 3.6).

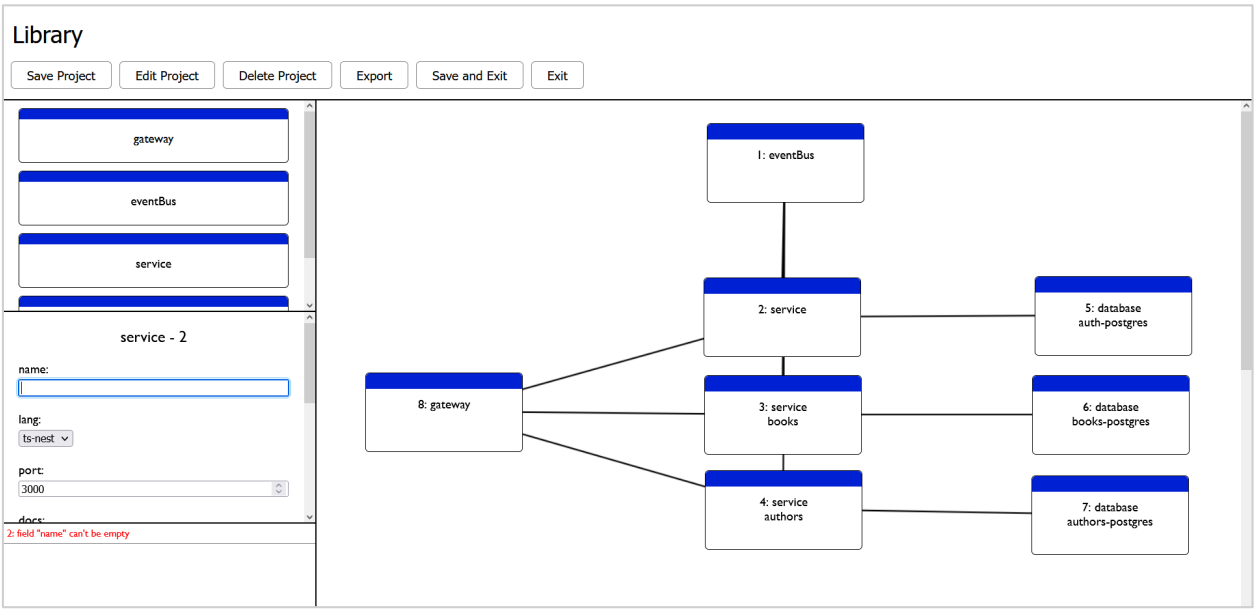


Рисунок 3.6 – Сторінка редагування проекту

У верхній частині сторінки (рисунок 3.7) можна змінити назву проекту, для цього треба натиснути на поточну назву проекту і ввести нову назву. Також тут знаходяться кнопки для управління проектом. За допомогою кнопки “Save Project” можна зберегти проект. Кнопка “Edit Project” відкриває панель налаштувань проекту (рисунок 3.8), де можна, наприклад, обрати систему CI/CD, яка буде використовуватись. За допомогою кнопки “Delete Project” можна видалити проект, після натискання на неї відкривається діалогове вікно (рисунок 3.9), в якому треба підтвердити бажання видалити проект. Кнопка “Export” дозволяє користувачу отримати конфігураційний файл з описом бажаного веб-додатку, для цього потрібно, щоб у проекті не було помилок. За допомогою кнопки “Save and Exit” можна зберегти проект і вийти, для того, щоб вийти без зберігання треба натиснути на кнопку “Exit”.

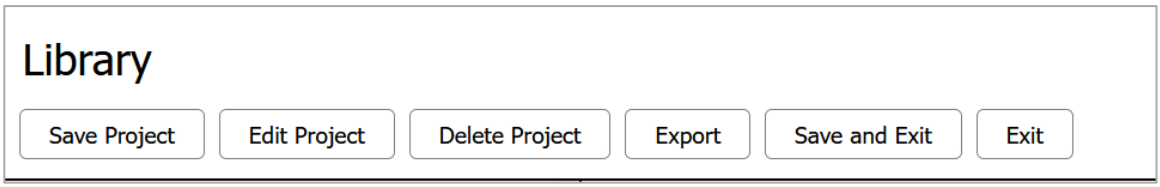


Рисунок 3.7 – Header сторінки редагування проекту

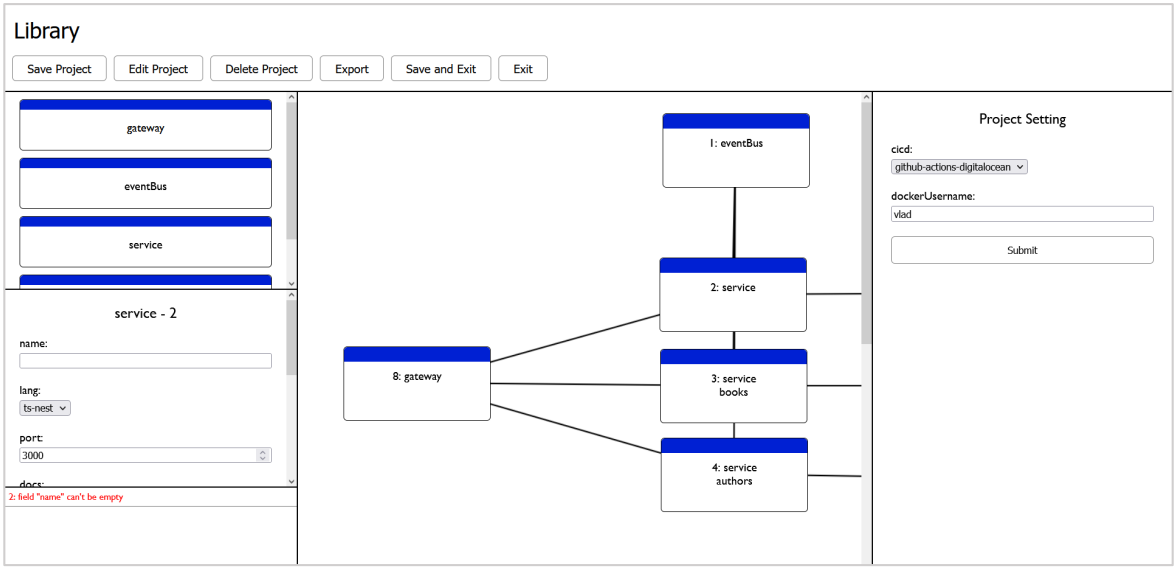


Рисунок 3.8 – Сторінка редагування проекту з відкритою панеллю налаштувань проекту

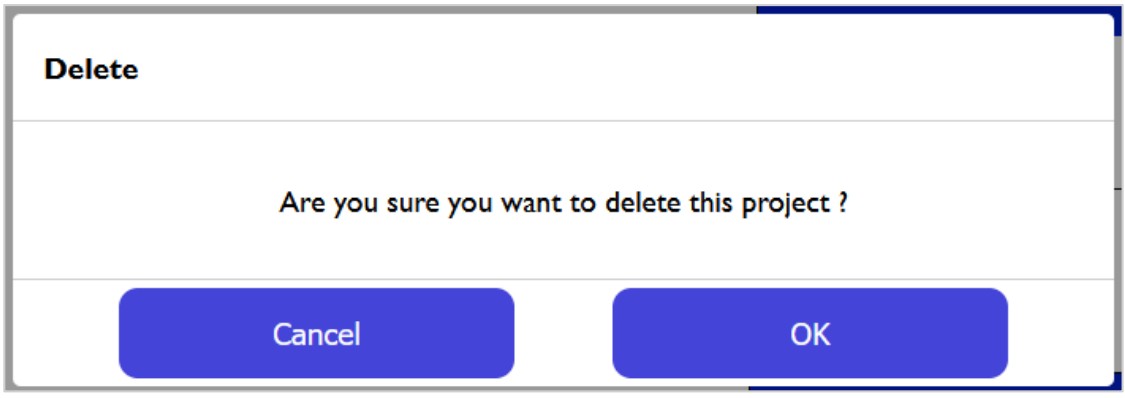


Рисунок 3.9 – Діалогове вікно підтвердження видалення проекту

На робочій області (рисунок 3.10) відображаються додані елементи: мікросервіси, бази даних, шлюзи API, брокери повідомлень.

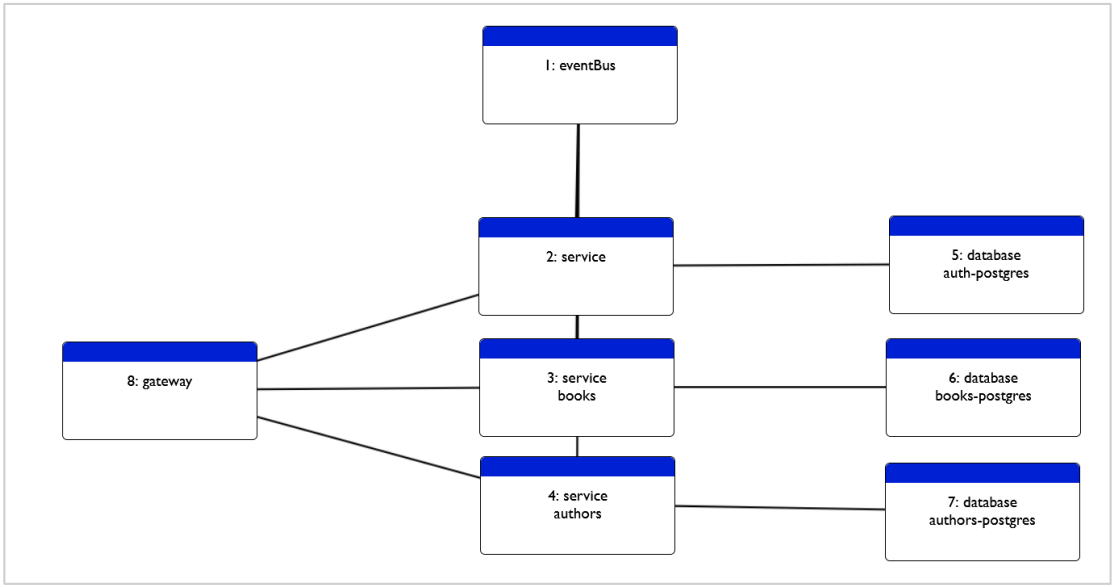


Рисунок 3.10 – Робоча область

Щоб додати елемент треба натиснути двічі на бажаний елемент на панелі елементів (рисунок 3.11).

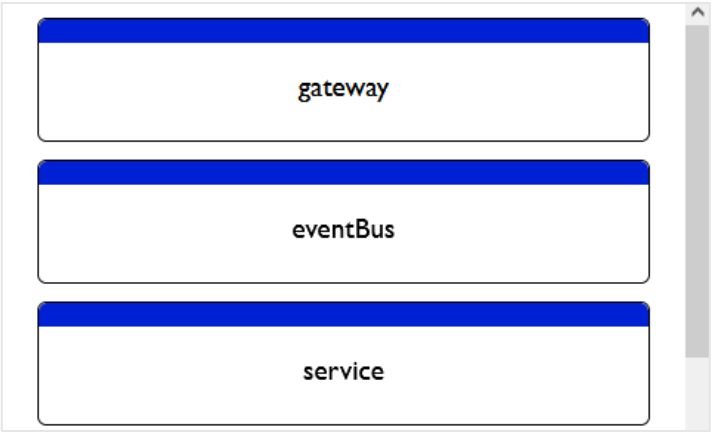


Рисунок 3.11 – Панель елементів

Для того, щоб змінити положення елементу на робочій області треба перетягнути його за верхню частину (має синій колір). Якщо ж натиснути на елемент, та не відпускаючи мишу, навести курсор на інший елемент і відпустити, то, якщо елементи сумісні, між ними буде створено зв’язок. Якщо ж елементи не є сумісними, то буде отримано повідомлення про помилку.

Якщо натиснути на елемент двічі, то його налаштування можна буде змінити на панелі редагування елементу (рисунок 3.12). Крім зміни налаштувань елементу на ній можна видалити елемент за допомогою кнопки “Delete” або ж видалити всі зв'язки елементу за допомогою кнопки “Remove All Connections”.

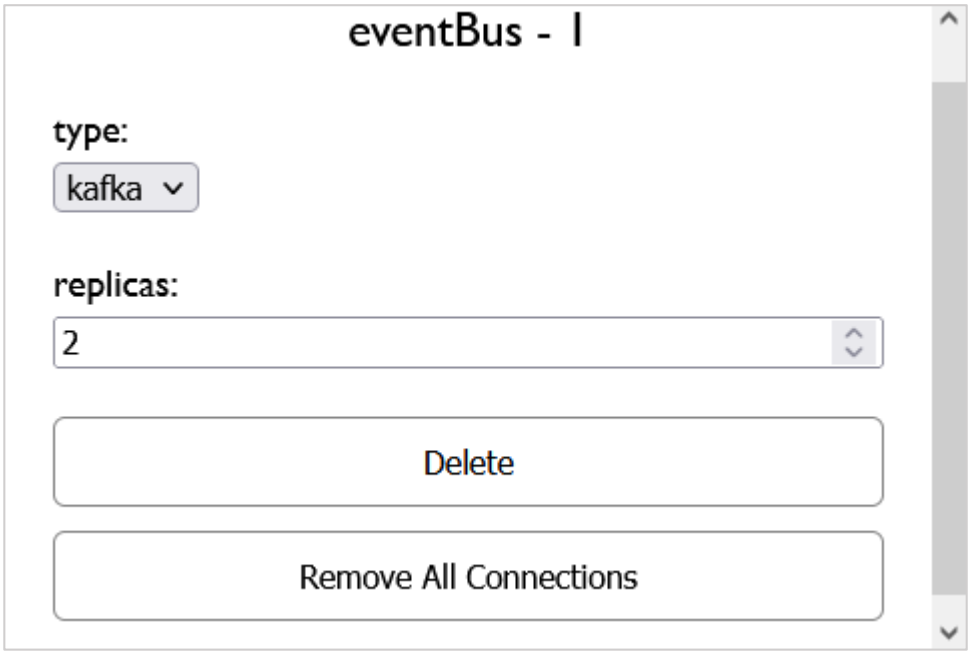


Рисунок 3.12 – Панель редагування елементу

На панелі помилок (рисунок 3.13) відображаються усі повідомлення про помилки у проекті. Такі повідомлення бувають двох типів: такі, що не зникають поки помилка не буде виправлена, і такі, що не потребують виправлення помилок і зникають після натискання кнопки “OK”.

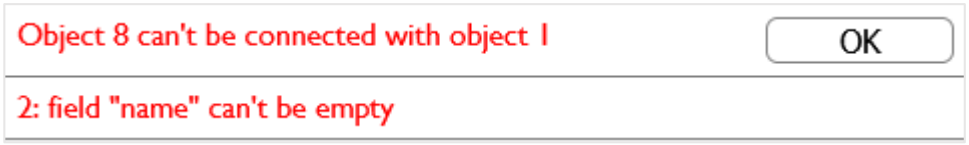


Рисунок 3.13 – Панель помилок

Після того, як усі бажані елементи були додані і налаштовані, необхідно натиснути на кнопку “Export”, для того, щоб отримати конфігураційний файл. Після цього треба виконати у консолі команду:

microservice-constructor –source=x –target=y,

де замість “x” вставити шлях до конфігураційного файлу, а замість “y” шлях для зберігання готового макету проекту. Після цього треба ввести дані за запитом додатку. Після цього згенерований макет додатку можна буде знайти за вказаним перед цим шляхом.

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2022 р.

Програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою

Схема структурна варіантів використання

КПІ.ІП-8321.045490.06.99

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Максим ГОЛОВЧЕНКО

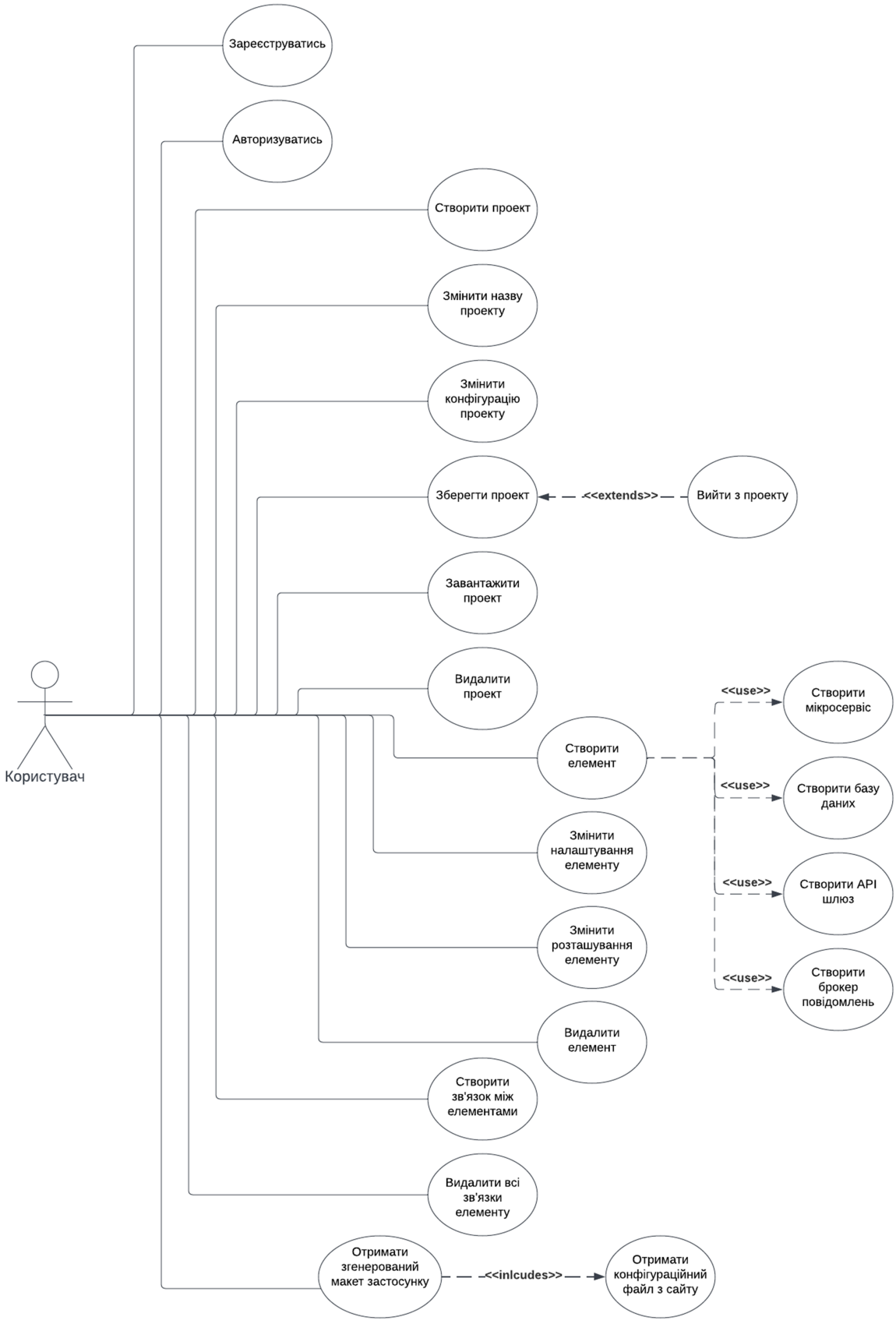
Нормоконтроль:

_____ Максим ГОЛОВЧЕНКО

Виконавець:

_____ Владислав ХМАРА

Київ – 2022



Зм.	Арк.	№ докум.	Підп.	Дата
Розроб.		Хмара В.В.		
Перев.		Головченко М.М		
Т. Кон.				
Н. Кон.		Головченко М.М		
Затв.		Головченко М.М		

КПІ.ІП-8321.045490.06.99.СС

Схема структурна варіантів використань

Лит.	Арк.	Аркушів
		1
Аркуш		Аркушів

Програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-81

Інв. № підл.	Підп. дата	Інв. № взам. підл.	Інв. № дубл.	Підп. дата

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2022 р.

Програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою

Схема бази даних

КПІ.ІП-8321.045490.07.99

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Максим ГОЛОВЧЕНКО

Нормоконтроль:

_____ Максим ГОЛОВЧЕНКО

Виконавець:

_____ Владислав ХМАРА

Київ – 2022

Зм.	Арк.	№ докум.	Підп.	Дата
Розроб.		Хмара В.В.		
Перев.		Головченко М.М		
Т. Кон.				
Н. Кон.		Головченко М.М		
Затв.		Головченко М.М		

КПІ.ІП-8321.045490.07.99.СБД

Схема бази даних

Лит.	Арк.	Аркушів
		1
Аркуш		Аркушів

Програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-81

Інв. № підп. | Підп. дата | Інв. № взам. підп. | Інв. № дубл. | Підп. дата

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2022 р.

Програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою

Креслення вигляду екранних форм

КПІ.ІП-8321.045490.08.99

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Максим ГОЛОВЧЕНКО

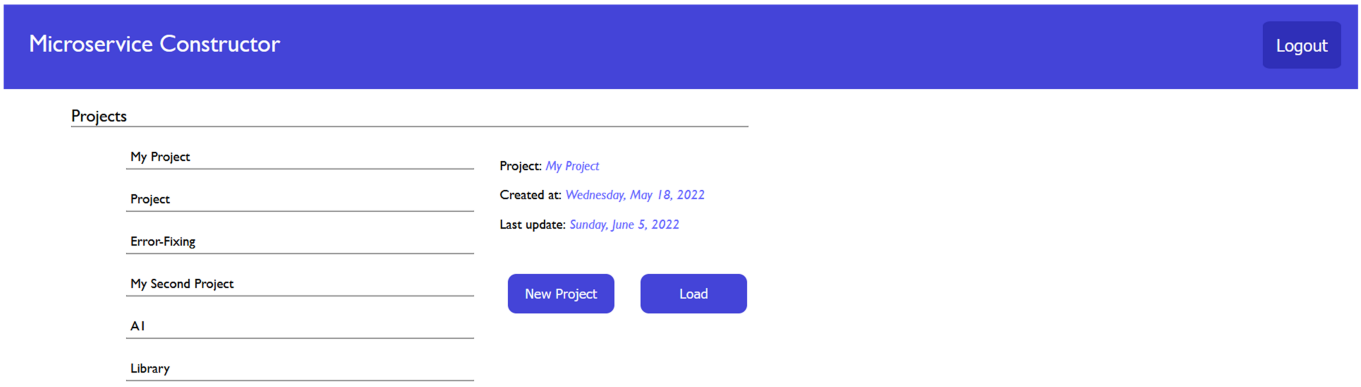
Нормоконтроль:

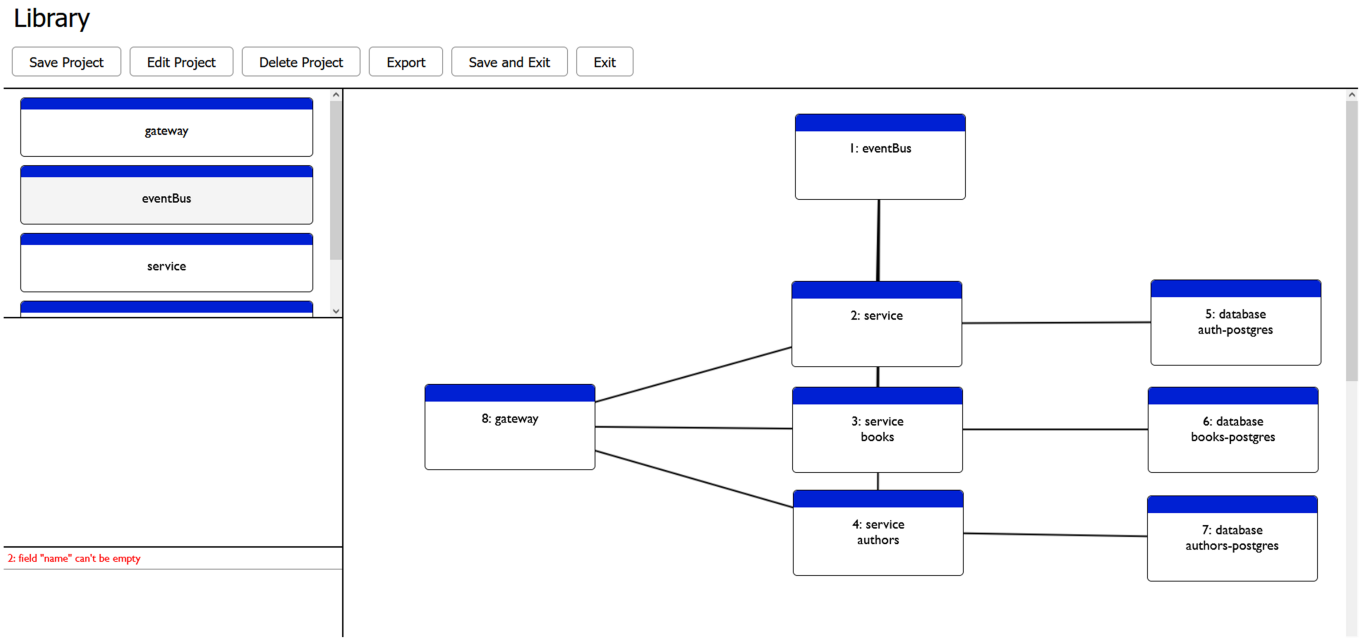
_____ Максим ГОЛОВЧЕНКО

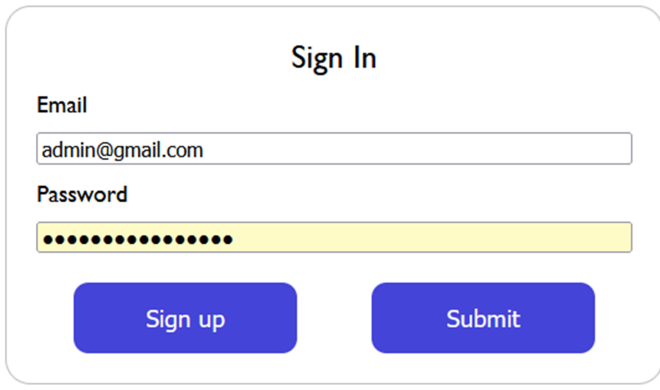
Виконавець:

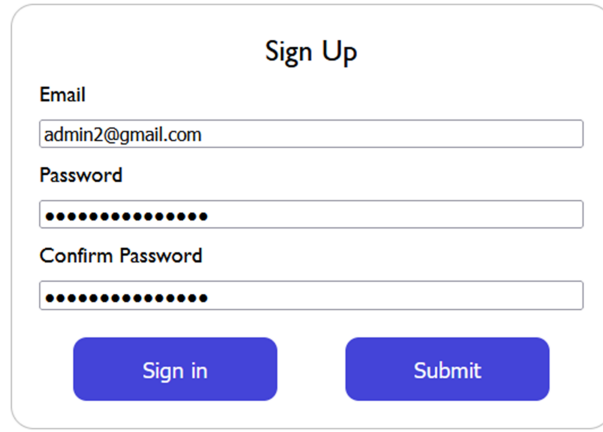
_____ Владислав ХМАРА

Київ – 2022









Зм.	Арк.	№ докум.	Підп.	Дата
Розроб.		Хмара В.В.		
Перев.		Головченко М.М		
Т. Кон.				
Н. Кон.		Головченко М.М		
Затв.		Головченко М.М		

КПІ.ІП-8321.045490.08.99.КЕ

Креслення вигляду екранних форм

Лит.	Арк.	Аркушів
		1
Аркуш		Аркушів

Програмне забезпечення для автоматизованого конструювання макетів веб-застосунків з мікросервісною архітектурою

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-81

Підп. дата | Інв. № дубл. | Інв. № взам. підп. | Підп. дата | Інв. № підп.