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

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

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

Дипломний проєкт

на здобуття ступеня бакалавра

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

спеціальності «121 Інженерія програмного забезпечення»

на тему: Веб-сервіс для генерації та розгортання лендингів

Виконав студент IV курсу, групи ІТ-93
(шифр групи)

Клепіков Станіслав Євгенович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник асистент каф. ІПІ Храмченко М. С _____
(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 Сперкач М. О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ – 2023

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

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

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

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

ЗАВДАННЯ
на дипломний проєкт студенту
Клепікову Станіславу Євгеновичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Веб-сервіс для генерації та розгортання лендингів
керівник проєкту Храмченко Микола Сергійович, каф. ІПІ, асистент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «_31_»квітня 2023 р. № _2101-с_

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

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметної область, огляд існуючих програмних рішень, постановка задачі, розроблення функціональних та нефункціональних вимог.
2) Моделювання та констуювання програмного забезпечення: моделювання бізнес-процесів, проектування архітектури та бази даних програмного забезпечення, аналіз безпеки даних.
3) Аналіз якості та тестування програмного забезпечення: аналіз якості кодової бази, тестування програмного забезпечення.
4) Впровадження та супровід програмного забезпечення: розгортання та підтримка програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань
2) Схема бази даних
3) Схема структурна класів програмного забезпечення

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «10» березня 2023 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	15.03.2023	
2	Аналіз існуючих методів розв’язання задачі	21.03.2023	
3	Постановка та формалізація задачі	25.03.2023	
4	Розробка інформаційного забезпечення	07.03.2023	
5	Алгоритмізація задачі	11.04.2023	
6	Обґрунтування вибору використаних технічних засобів	13.04.2023	
7	Розробка програмного забезпечення	08.05.2023	
8	Налагодження програми	15.05.2023	
9	Виконання графічних документів	20.05.2023	
10	Оформлення пояснювальної записки	29.05.2023	
11	Подання ДП на попередній захист	30.05.2023	
12	Подання ДП рецензенту		
13	Подання ДП на основний захист		

Студент _____ Станіслав КЛЕПІКОВ
(підпис) (ініціали, прізвище)

Керівник _____ Микола ХРАМЧЕНКО
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 24 таблиць, 18 рисунків та 21 джерел – загалом 68 сторінок.

Дипломний проєкт присвячений розробці веб-сервісу для генерації та розгортання лендингів.

Метою дипломного проекту є спрощення та прискорення процесу створення та розгортання лендингів з метою вирішення проблеми складності і затрат, пов'язаних з цими процесами.

Об'єкт дослідження: розробка веб-сервісу для генерації та розгортання лендингів.

Предмет дослідження: веб-сервіс для генерації та розгортання лендингів, його функціональні можливості, технології та алгоритми розробки.

У розділі аналізу вимог до програмного забезпечення було ретельно розглянуто предметну область та сформульовано функціональні та нефункціональні вимоги до проєкту.

У розділі моделювання та конструювання програмного забезпечення було виконано наступні дії: визначено бізнес-процеси, розроблена архітектура програмного забезпечення та структура бази даних, а також проведений аналіз безпеки даних продукту.

У розділі аналізу якості та тестування програмного забезпечення було проведено оцінку якості кодової бази та здійснено комплексне тестування всього програмного забезпечення.

В розділі впровадження та супровід програмного забезпечення було проведено ефективне впровадження та підтримку веб-сервісу, використовуючи платформу Heroku.

КЛЮЧОВІ СЛОВА: ВЕБ-СЕРВІС, ВЕБ-СТОРІНКА, ЛЕНДИНГ, ШАБЛОН, БІЗНЕС, СТВОРЕННЯ, РОЗГОРТАННЯ, БАЗА ДАНИХ, АРХІТЕКТУРА.

ABSTRACT

The explanatory note of the diploma project consists of four sections and includes 24 tables, 18 figures, and 21 references, totaling 68 pages.

The diploma project is dedicated to the development of a web service for generating and deploying landing pages.

The aim of the diploma project is to simplify and expedite the process of creating and deploying landing pages, addressing the challenges of complexity and costs associated with these processes.

The research object is the development of a web service for generating and deploying landing pages.

The research subject is the web service for generating and deploying landing pages, its functional capabilities, development technologies, and algorithms.

In the section on software requirements analysis, the subject area was thoroughly examined, and functional and non-functional requirements for the project were formulated.

In the section on software modeling and construction, the following actions were performed: identification of business processes, development of software architecture and database structure, and analysis of data security for the product.

In the section on quality analysis and software testing, an evaluation of the code quality was conducted, and comprehensive testing of the entire software was performed.

In the section on implementation and support of the software, effective deployment and maintenance of the web service were carried out using the Heroku platform.

KEYWORDS: WEB SERVICE, WEB PAGE, LANDING, TEMPLATE, BUSINESS, CREATION, DEPLOYMENT, DATABASE, ARCHITECTURE.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-9314.045440.01.91	Технічне завдання	17	
3	А4	КПІ.ІТ-9304.045440.02.81	Пояснювальна записка	67	
4	А4	КПІ.ІТ-9314.045440.03.12	Текст програми	42	
5	А4	КПІ.ІТ-9314.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІТ-9314.045440.05.34	Керівництво користувача	9	
7	А3	КПІ.ІТ-9314.045440.06.99.СС	Схема структурна варіантів використань	1	
8	А3	КПІ.ІТ-9314.045440.07.99.СБД	Схема бази даних	1	
9	А3	КПІ.ІТ-9314.045440.08.99.СС	Схема структурна класів програмного забезпечення	1	

					КПІ.ІТ-9314.045440.01.90		
Змін.	Арк.	№ докум.	Підп.	Дата			
Розроб.		Клепіков С. Є.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Храмченко. М. С.					
Н.контр.		Головченко М. М.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-93	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2023 р.

ВЕБ-СЕРВІС ДЛЯ ГЕНЕРАЦІЇ ТА РОЗГОРТАННЯ ЛЕНДИНГІВ

Технічне завдання

КПІ.ІТ-9314.045440.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Микола ХРАМЧЕНКО

Нормоконтроль:

_____ Максим ГОЛОВЧЕНКО

Виконавець:

_____ Станіслав КЛЕПІКОВ

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-сервіс для генерації та розгортання лендингів

Галузь застосування: Наведене технічне завдання поширюється на розробку програмного забезпечення «Веб-сервіс для генерації та розгортання лендингів» КПІ.ІТ-9314.045440.01.91, котра використовується для створення та редагування лендингів та призначена для підприємців та власників бізнесу.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки веб-сервісу для генерації та розгортання лендингів 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 створення та редагування лендингів і може використовуватись для рекламних кампаній, збіру потенційних клієнтів та продажу продуктів або послуг.

Метою розробки є спрощення процесу створення лендингів, забезпечення його доступності та швидкості. Цей сервіс дозволяє навіть користувачам без технічних навичок легко створювати та налаштовувати власні лендинги, використовуючи готові шаблони та інтуїтивно зрозумілий інтерфейс.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– Форма реєстрації користувача (Рисунок 4.1);

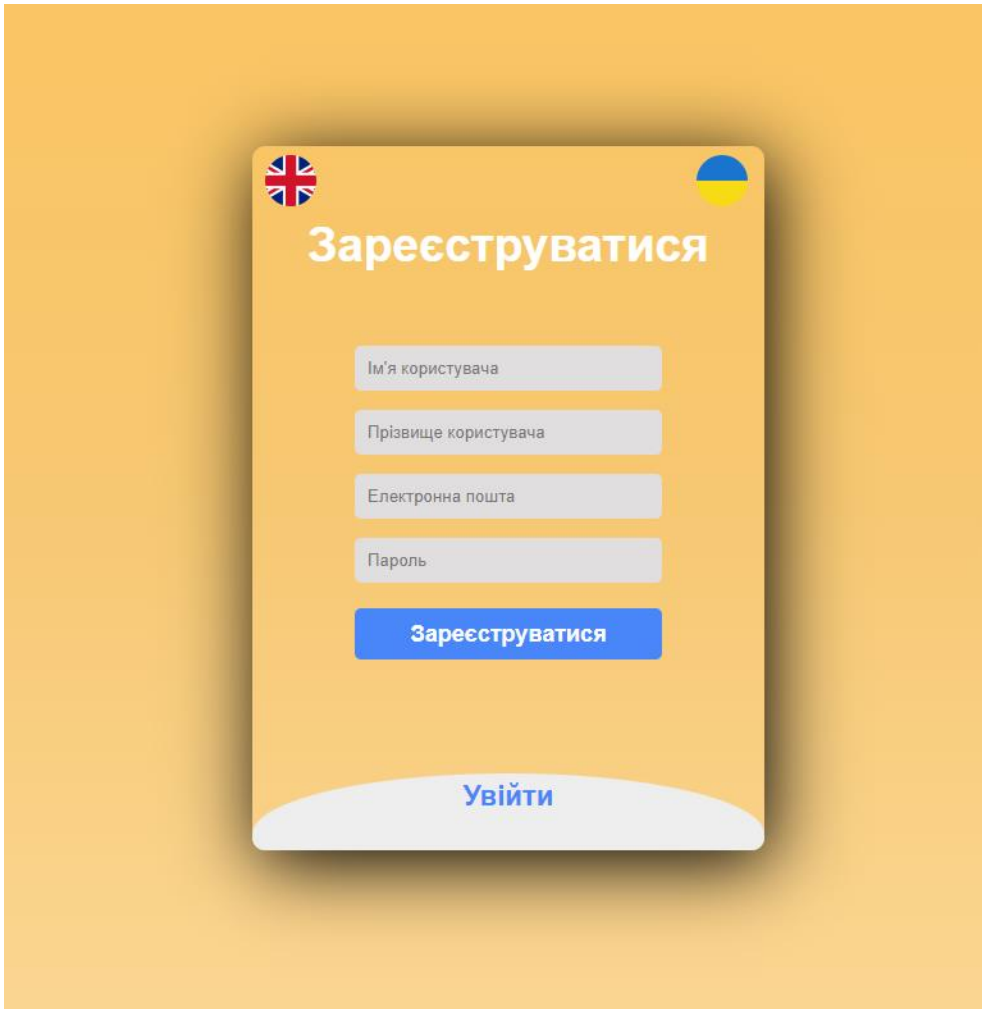


Рисунок 4.1 – Форма реєстрації користувача

– При введенні некоректних даних в відповідні текстові поля, повинні відображатись відповідні повідомлення (Рисунок 4.2);

Рисунок 4.2 – Повідомлення про некоректно введені значення

– Форма авторизації (Рисунок 4.3);

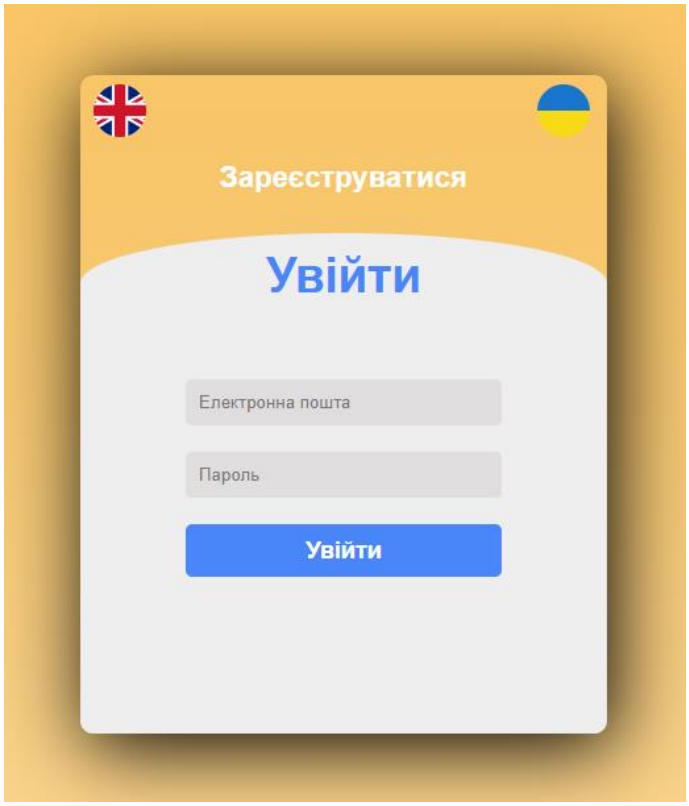


Рисунок 4.3 – Форма авторизації

– При введені некоректних значень в поля адреси електронної пошти та паролю, повинні з’являтись з відповідні повідомленням (Рисунок 4.4);

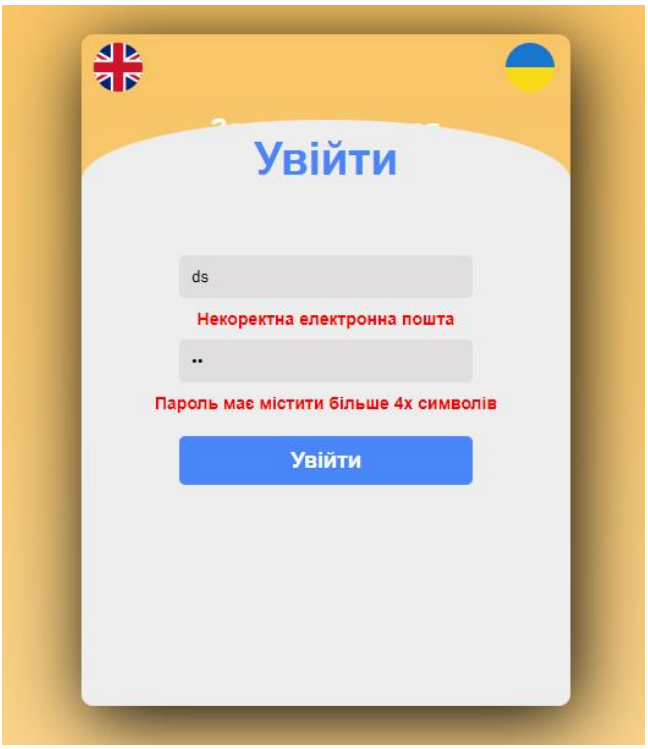


Рисунок 4.4 – Повідомлення про некоректні дані

– При введенні коректних значень адреси електронної пошти та паролю, користувач повинен перенаправлятись на домашню сторінку (Рисунок 4.5);

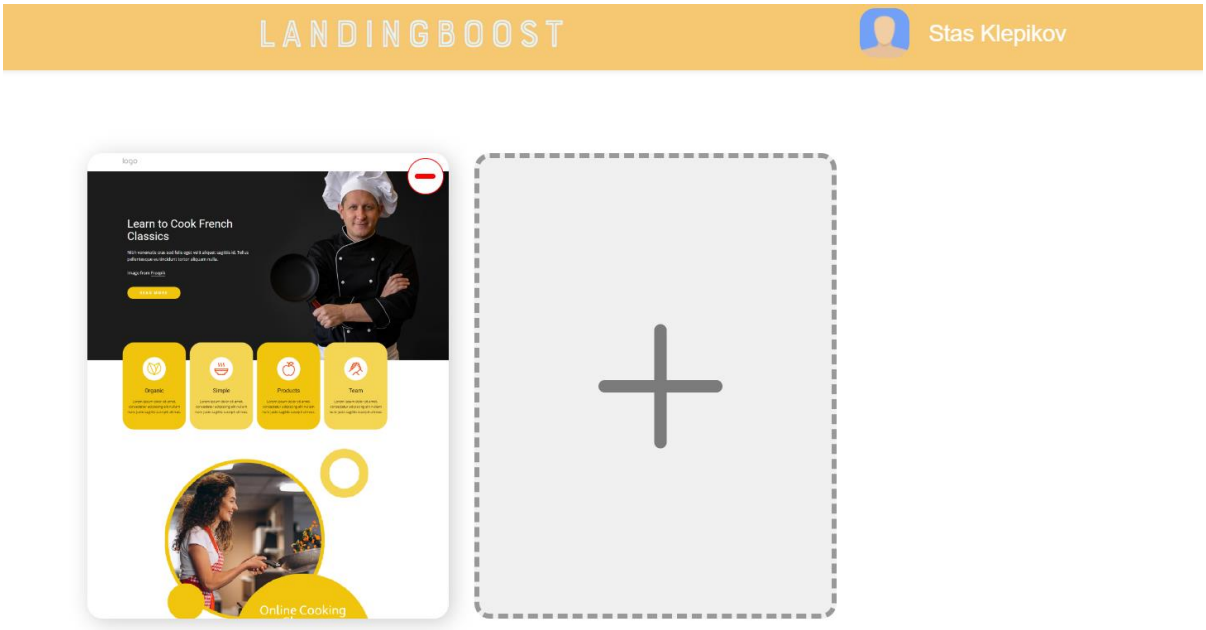


Рисунок 4.5 – Домашня сторінка користувача

– При натисканні на кнопку «+» (Рисунок 4.5) повинно відкриватися модальне вікно зі списком шаблонів (Рисунок 4.6);

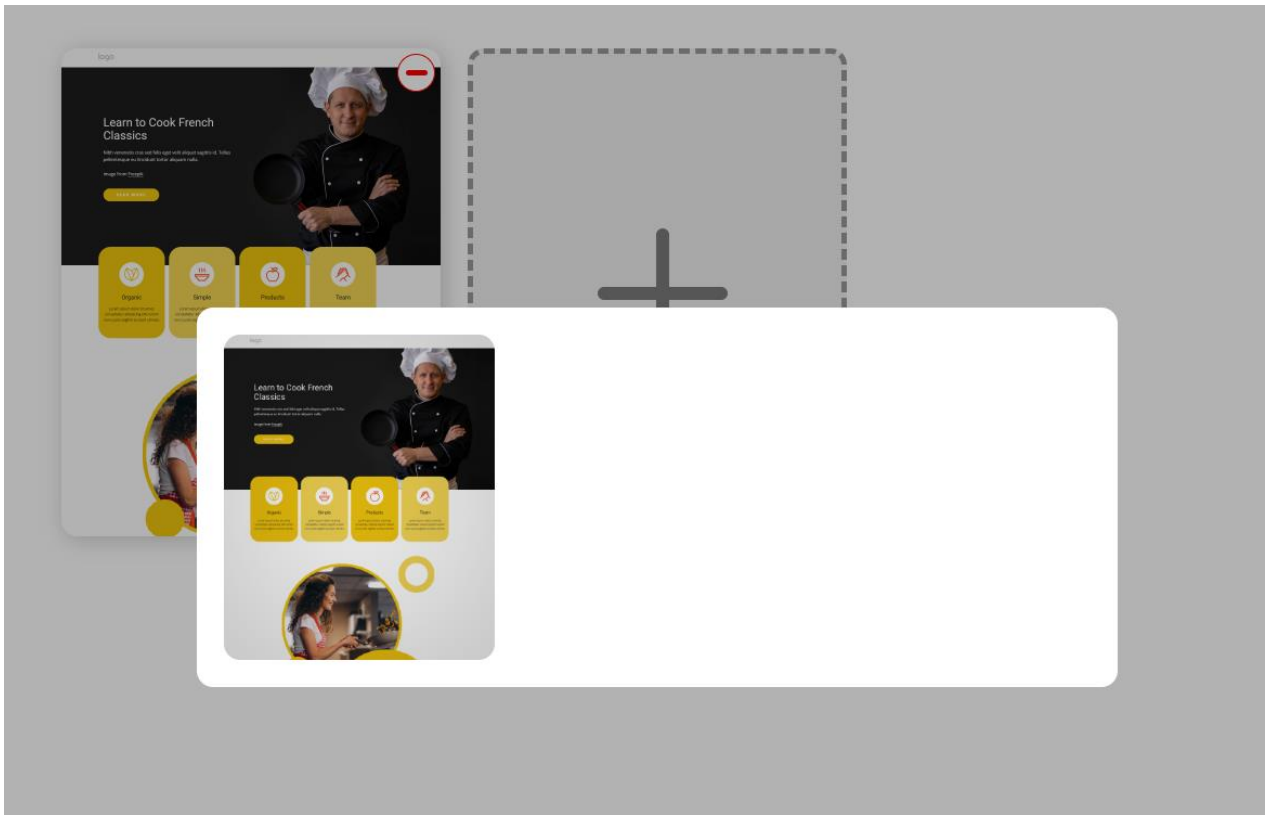


Рисунок 4.6 – Модальне вікно зі списком шаблонів

– Після вибору необхідного шаблону (Рисунок 4.6), повинна створитися картка цього шаблона на домашній сторінці (Рисунок 4.7);

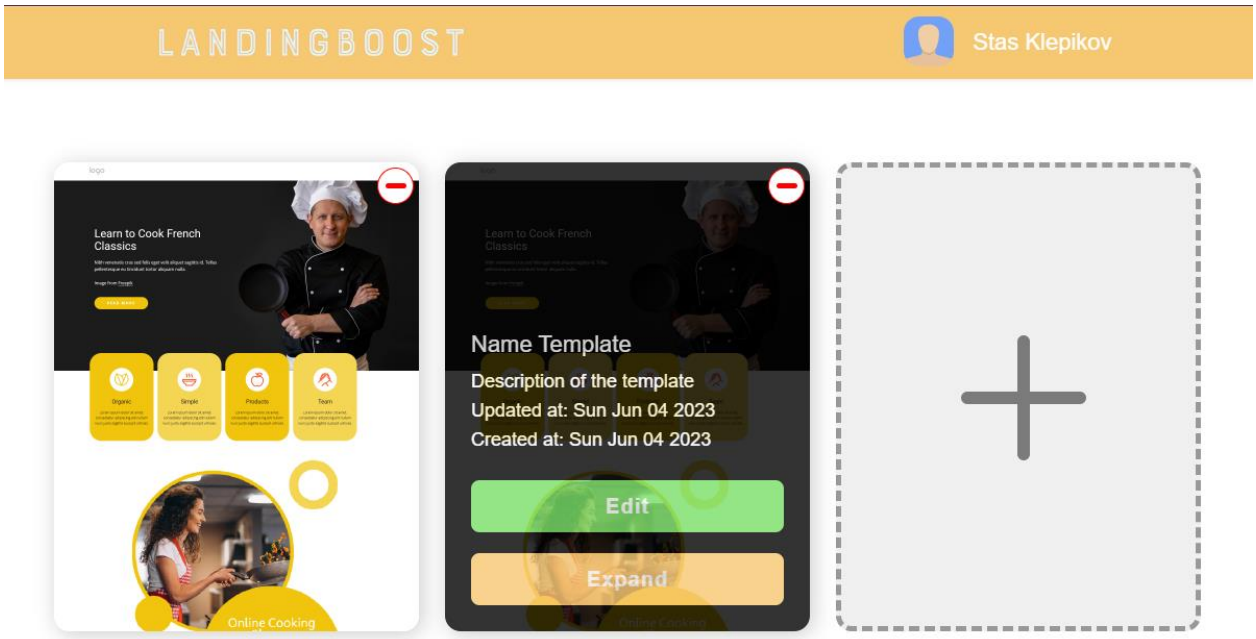


Рисунок 4.7 – Картка шаблону на домашній сторінці

– При натисканні кнопки «Редагувати» користувача перенаправляє на веб-сторінку шаблону в режимі редагування, де він може змінювати наповнення веб-сторінки та кольорову гаму. Відредагувавши шаблон, користувач натискає кнопку «Зберегти», щоб зберегти зміни в шаблоні (Рисунок 4.8);

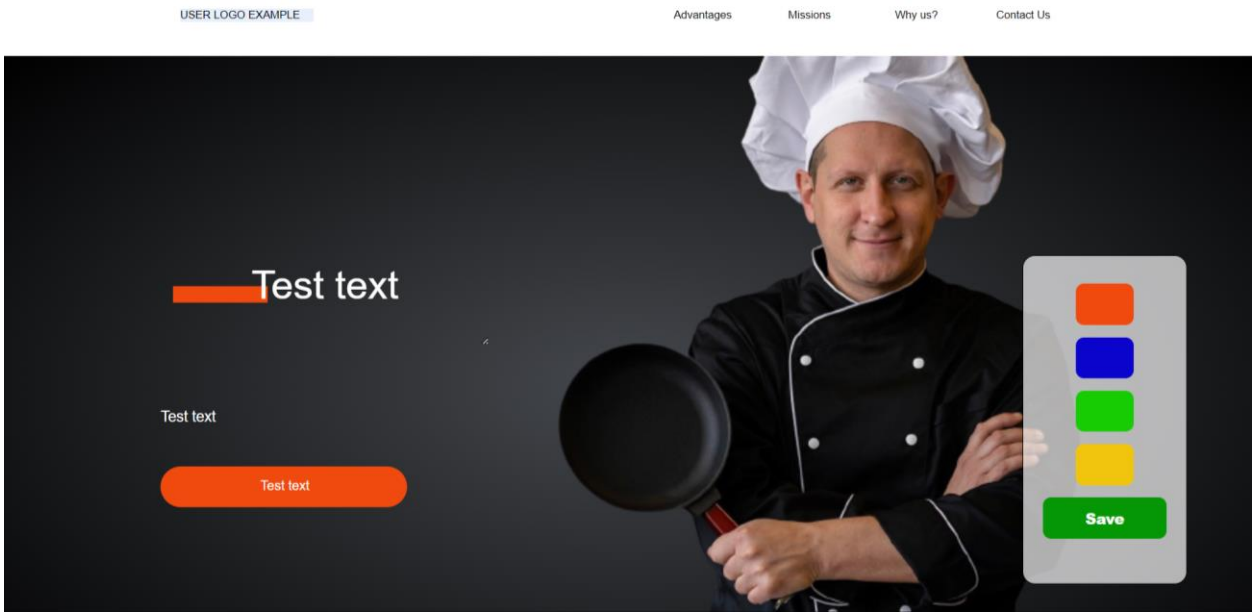


Рисунок 4.8 – Веб-сторінка в режимі редагування

– Після внесення змін в шаблон, користувач може розгорнути веб-сторінку та подивитися збережені зміни (Рисунок 4.9);

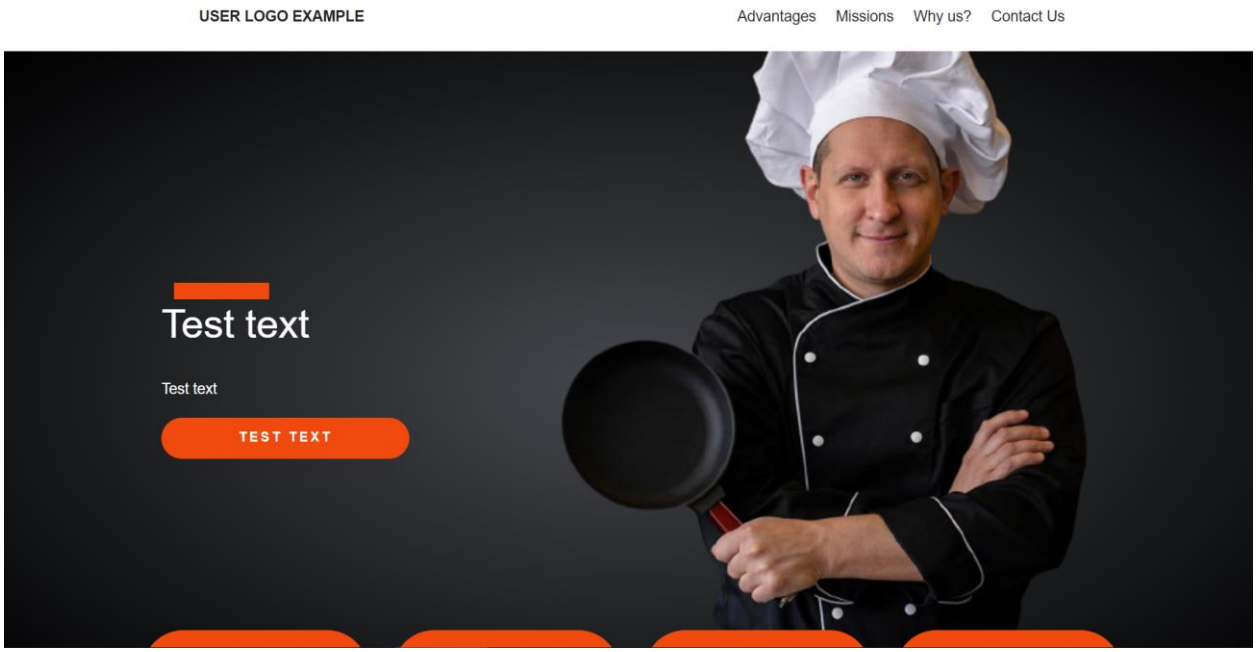


Рисунок 4.9 – Розгорнута веб-сторінка з внесеними змінами

– Якщо користувачу потрібно видалити веб-сторінку, він може натиснути кнопку «-» в верхньому правому куті та веб-сторінка зникне з домашньої сторінки користувача та видалиться (Рисунок 4.10);

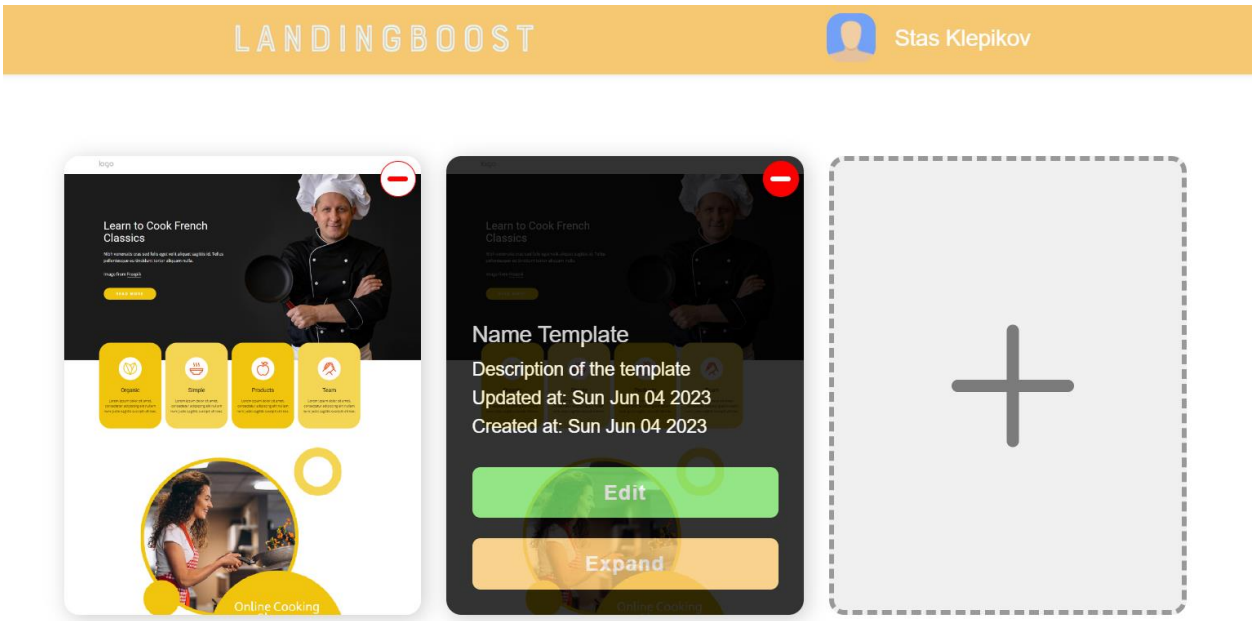


Рисунок 4.10 – Видалення веб-сторінки

4.1.2 Для неавторизованого користувача:

– Реєстрація користувача шляхом введення імені, прізвища, адреси електронної пошти та паролю;

– Перевірка імені та прізвища користувача на довжину від 2 до 64 символів;

– Перевірка адреси електронної пошти на відповідність шаблону <username>@<domain>, де <username> - унікальне ім’я користувача, а <domain> - домен, до якого прив’язана електронна пошта;

– Перевірка паролю на відповідність наступним вимогам:

– Довжина від 4 до 64 символів;

– Авторизація шляхом введення адреси електронної пошти та паролю;

4.1.3 Для авторизованого користувача:

– Переглянути список створених веб-сторінок;

– Створити веб-сторінку на основі шаблону;

– Перейти в режим редагування веб-сторінки;
– Змінення вмісту сторінки;
– Збереження змін в веб-сторіннці;
– Розгортання веб-сторінки;
– Видалення веб-сторінки;
– Вихід з облікового запису;

4.1.4 Для адміністратора:

В проєкті наявність адміністратора не передбачено.

4.2 Вимоги до надійності

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

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на комп'ютерах із встановленим браузером та доступом до Інтернету.

Мінімальна конфігурація технічних засобів:

тип процесору: Intel Core i3;

об'єм ОЗП: 4 Гб.

Рекомендована конфігурація технічних засобів:

тип процесору: Intel Core i5;

об‘єм ОЗП: 16 Гб.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно бути сумісним з будь-яким сучасним веб-браузером, незалежно від операційної системи, на якій він використовується.

4.5.1 Вимоги до вхідних даних

В проєкті не встановлені обов'язкові вимоги до вхідних даних.

4.5.2 Вимоги до вихідних даних

В проєкті не встановлені обов'язкові вимоги до вихідних даних.

4.5.3 Вимоги до мови розробки

Розробку виконати на мовах програмування JavaScript.

4.5.4 Вимоги до середовища розробки

Розробку виконати за допомогою середовища Visual Studio Code.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді текстових файлів.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Для розгортання програмного забезпечення передбачається використання сервісу Heroku.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

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

– пояснювальна записка;

– технічне завдання;

– текст програми;

– програма та методика тестування;

– керівництво користувача.

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

– схема структурна варіантів використання;

– схема бази даних;

– схема структурна класів програмного забезпечення.

5.2 Спеціальні вимоги до програмної документації

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

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення рекомендованої літератури	15.03	
2.	Аналіз існуючих методів розв’язання задачі	21.03	
3.	Постановка та формалізація задачі	25.03	Мета розробки
4.	Розробка інформаційного забезпечення	07.03	Схема архітектури, компонентів програмного забезпечення та діаграма класів
5.	Алгоритмізація задачі	11.04	Фізична модель бази даних
6.	Обґрунтування вибору використаних технічних засобів	13.04	
7.	Розробка програмного забезпечення	08.05	Тексти програмного забезпечення
8.	Налагодження програми	15.05	Тести, результати тестування
9.	Виконання графічних документів	20.05	Графічний матеріал проекту
10.	Оформлення пояснювальної записки	29.05	Пояснювальна записка

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

Пояснювальна записка

до дипломного проєкту

на тему: Веб-сервіс для генерації та розгортання лендингів

КПІ.ІТ-9314.045440.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AWS – Amazon Web Services – платформа хмарних обчислень

BPMN – Business Process Management Initiative – система умовних позначень для моделювання бізнес-процесів

CI/CD – Continuous Integration / Continuous Deployment – методика розробки програмного забезпечення

CSP – Content Security Policy – стандарт захисту сайтів від атак із впровадженням контенту

CSS – Cascading Style Sheets – формальна мова опису зовнішнього вигляду документа

DOM – Document Object Model – специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами.

FTP – File Transfer Protocol – протокол передачі файлів через мережу

HTML – Hypertext Markup Language – стандартизована мова розмітки документів для перегляду вебсторінок у браузері

MVC – Model-View-Controller – архітектурний шаблон

ORDBMS – Object-Relational Database Management System – система керування базами даних

Redis – Remote Dictionary Server – розподілене сховище пар ключ-значення, які зберігаються в оперативній пам'яті

SCSS – Syntactically Awesome Stylesheets – це метамова на основі CSS, призначена для збільшення рівня абстракції CSS-коду

WYSIWYG – What You See Is What You Get – тип інтерфейсу користувача, який дозволяє редагувати документи або створювати вміст

ВСТУП

Веб-сервіси для генерації та розгортання лендингів стають все більш актуальними у сучасному світі. Швидкість і зручність створення лендингів відчутно полегшує процес реклами та просування товарів та послуг в інтернеті.

Цей проєкт є відповіддю на підвищену потребу у веб-сервісах для автоматизації процесу створення та розгортання лендингів. Задачею даного проєкту є розробка інструменту, що дозволяє створювати лендинги без необхідності великої кількості знань та досвіду веб-програмування.

Провідні наукові установи та організації, а також провідні вчені та фахівці певних галузей досліджували та вивчали процес створення лендингів, зокрема застосування веб-сервісів для цієї мети. Проте, на сьогоднішній день, багато інструментів для створення лендингів мають обмежену функціональність та потребують великої кількості часу та знань для їх використання.

Метою даного проєкту є спрощення та прискорення процесу створення та розгортання лендингів з метою вирішення проблеми складності і затрат, пов'язаних з цими процесами. Веб-сервіс прагне надати користувачам простий та ефективний інструмент, що дозволить їм успішно просувати свої інфопродукти або послуги в онлайн-середовищі, економлячи при цьому час та зусилля.

Можливі сфери застосування веб-сервісу для генерації та розгортання лендингів включають також електронну комерцію, а також інтернет-магазини та інші онлайн-бізнеси. Веб-сервіс може бути використаний для швидкої та ефективної створення лендінгів для різних товарів та послуг, що дозволяє прискорити процес продажу та підвищити конверсію.

Крім того, веб-сервіс може бути використаний для створення лендінгів з метою збору даних від користувачів, наприклад, для проведення опитувань або реєстрації на події.

У сфері освіти та тренінгів веб-сервіс може бути використаний для створення лендінгів з метою реклами курсів та тренінгів, а також для збору даних про потенційних учасників.

Також веб-сервіс може бути корисним для стартапів та малих бізнесів, які хочуть швидко запустити свою присутність в Інтернеті і привернути нових клієнтів.

Отже, веб-сервіс для генерації та розгортання лендингів має широкі можливості застосування в різних галузях та сферах бізнесу.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

В сучасному світі більшість бізнесів діють в Інтернеті, тому мають веб-сайти, які виконують різноманітні функції, такі як продаж товарів або послуг, надання інформації про компанію, збір контактних даних потенційних клієнтів тощо. Однак, щоб залучити нових клієнтів, підвищити конверсію і збільшити продажі, можна використовувати спеціальні сторінки – лендінги.

Дослідження авторів Bhatia, Jain та Singh «The Role of Landing Pages in Online Marketing» показало, що лендінги можуть збільшити конверсію відвідувачів сайту в покупців. Крім того, автори відзначають, що ефективність лендінгу залежить від правильної розробки та оформлення.

Лендінг – це сторінка веб-сайту, створена з метою привернути увагу потенційних клієнтів та перетворити їх на покупців, оформивши замовлення на продукт або послугу. Лендінги можуть містити інформацію про товар або послугу, переваги придбання в даній компанії, відгуки клієнтів, а також форму для збору контактних даних[1].

Основна мета створення веб-сервісу полягає в тому, щоб покращити процес створення лендінгів, обираючи зі списку готові шаблони, редагуючи їх вміст та оформлення з використанням зручного інтерфейсу, а також миттєво розгортати їх на сервері. Використання веб-сервісу для створення лендінгів дозволяє користувачам створювати професійні сторінки без необхідності мати технічні знання, працюючи з візуальним інтерфейсом та готовими шаблонами.

Для розробки веб-сервісу використовуватимуться сучасні веб-технології та фреймворки, що дозволяють ефективно створювати веб-додатки та забезпечувати їх безпеку та швидкодію. Окрему увагу буде приділено забезпеченню безпеки веб-сервісу та забезпеченню конфіденційності[2].

Для розгортання лендінгів необхідна інфраструктура, що дозволяє швидко публікувати сторінки в Інтернеті. Одним із можливих рішень є використання хмарних сервісів, які надають можливість зберігати і розгортати веб-сайти в Інтернеті.

Один з популярних хмарних сервісів – Amazon Web Services (AWS), який містить в собі такі послуги, як Amazon Elastic Compute Cloud (Amazon EC2), Amazon Simple Storage Service (Amazon S3) та Amazon Elastic Block Store (Amazon EBS). Amazon EC2 дозволяє запускати віртуальні машини з налаштуванням, яке задається користувачем[3]. Amazon S3 дозволяє зберігати дані, такі як зображення, CSS-файли, JavaScript-скрипти тощо. Amazon EBS забезпечує можливість зберігати та обробляти великі об’єми даних.

Також існують інші хмарні сервіси, які можуть бути використані для розгортання лендінгів, такі як Microsoft Azure, Google Cloud Platform, DigitalOcean тощо.

Для розробки веб-сервісу для генерації та розгортання лендінгів необхідно провести аналіз можливостей різних хмарних сервісів та вибрати оптимальний сервіс, що задовольняє вимогам проєкту. Також потрібно визначити, які технології будуть використані для розробки веб-сервісу та як будуть забезпечені потреби користувачів у зручному інтерфейсі для створення та редагування лендінгів.

1.2 Змістовний опис і аналіз предметної області

В галузі IT, що стосується веб-сервісів для генерації та розгортання лендингів, можна виділити наступні проблеми поточного стану:

– обмежена функціональність. Деякі наявні веб-сервіси мають обмежені можливості у створенні та налаштуванні лендингів. Це обмежує творчість та гнучкість при розробці унікальних та привабливих лендингів для клієнтів.

– складність використання. Деякі існуючі рішення можуть бути складними у використанні, вимагати глибоких технічних знань або велику кількість кроків для створення та розгортання лендингів. Це ускладнює роботу з ними для користувачів без достатнього досвіду.

– вартість та доступність. Деякі рішення дорого коштують або мають платні плани, що обмежує доступність для маленьких бізнесів або початківців. Це ставить під загрозу їх можливість використання таких сервісів з обмеженими бюджетами.

– безпека. Враховуючи зростання кіберзагроз, деякі веб-сервіси можуть мати недоліки у забезпеченні безпеки, такі як недостатня захищеність від атак або незадовільне керування доступом до даних. Це може підірвати довіру користувачів та призвести до втрати даних або порушення конфіденційності.

Ці недоліки підкреслюють потребу у подальшому розвитку та вдосконаленні веб-сервісів для генерації та розгортання лендингів, зокрема шляхом розробки більш функціональних, простих у використанні та доступних рішень.

Одним з основних аспектів, який потребує уваги, є спрощення використання веб-сервісу для генерації та розгортання лендингів. Це можна досягти шляхом розробки інтуїтивно зрозумілих і легко використовуваних інтерфейсів користувача, які дозволять навіть користувачам без глибоких технічних знань ефективно використовувати сервіс.

Забезпечення простоти використання включає в себе розробку логічних та зрозумілих кроків для створення та налаштування лендингів, чіткі пояснення та допомогу на кожному етапі процесу. Інтуїтивний інтерфейс користувача має включати інтерактивні елементи та простие навігаційне меню.

Важливо також враховувати потреби різних категорій користувачів і забезпечити гнучкі налаштування та персоналізацію. Для цього можна надати можливість вибору шаблонів, стилів, кольорів та інших елементів

дизайну лендингу. Це дозволить користувачам створювати унікальні та привабливі лендинги, враховуючи їх власні потреби і брендовий стиль.

Такий підхід до розробки інтерфейсу користувача сприятиме зниженню бар'єрів для використання веб-сервісу та забезпечить більш широкий доступ до цих можливостей навіть для тих, хто не має глибоких технічних знань. Таким чином, веб-сервіс стане більш привабливим і користувачі зможуть легко та ефективно створювати лендингі[4].

1.3 Аналіз існуючих технологій та успішних ІТ-проєктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації веб-сервісу для генерації та розгортання лендингів. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

В рамках дослідження було розглянуто різноманітні алгоритми та технічні підходи з метою вибору найбільш ефективних та оптимальних рішень для успішної реалізації проєкту.

Розглянемо алгоритми для генерації лендингів. Першими варто згадати шаблон-орієнтовані алгоритми, такі як:

– застосування готових шаблонів для створення лендингів. Цей підхід дозволяє швидко створювати лендинги за допомогою готових шаблонів, що значно економить час і зусилля розробника. Крім того, використання шаблонів може забезпечити професійний та стилізований вигляд лендингу без необхідності великої кількості дизайнерських робіт.

– реалізація шаблонів, які можуть змінюватися залежно від вхідних параметрів або даних. Цей підхід надає більшу гнучкість та можливість налаштування лендингів під конкретні потреби. Крім того, динамічні шаблони можуть забезпечувати персоналізацію контенту для користувачів, що покращує їх досвід.

Крім цього, згадаємо про наявність інтерактивних редакторів, як наприклад:

– WYSIWYG-редактори (What You See Is What You Get): Редактори, що дозволяють користувачам створювати лендинги шляхом простого перетягування та редагування елементів на сторінці. Вони надають миттєвий візуальний зв'язок між редагуванням та відображенням результату, що спрощує розробку лендингів навіть для користувачів без технічних навичок[5].

– Drag and Drop (Перетягування та розміщення): Цей підхід дає можливість користувачам перетягувати та розміщувати елементи для створення лендингу без потреби в ручному кодуванні. Він забезпечує простий та інтуїтивно зрозумілий інтерфейс, що полегшує процес створення лендингів тим, хто не володіє програмуванням[6].

Кожен з наведених алгоритмів та підходів має свої вигоди, і вибір залежить від конкретних вимог та потреб проєкту. В даному проєкті було обрано алгоритм "використання готових шаблонів", який забезпечує швидке створення лендингів з професійним дизайном. Цей алгоритм дозволяє ефективно використовувати наявні шаблони і забезпечує швидкість розробки без необхідності починати з нуля. Результатом є створення лендингів, які мають привабливий вигляд з мінімальними зусиллями.

Розглянемо технічні рішення для розгортання лендінгів, включаючи різні алгоритми та підходи.

– використання протоколу FTP для передачі файлів та розгортання лендингу на веб-сервері. Цей підхід вимагає наявності доступу до FTP-сервера та відповідних прав доступу. FTP-розгортання є простим і зручним, але вимагає ручного копіювання та оновлення файлів.

– використання системи контролю версій Git для керування та автоматичного розгортання лендингів на веб-сервері. Git-розгортання дозволяє зберігати історію змін, працювати з різними гілками розробки та ефективно керувати розгортанням лендингу, алє використання Git-

розгортання вимагає розуміння системи контролю версій Git та знання командного рядка. Це може бути перешкодою для користувачів без технічних навичок.

– використання хмарних сервісів для розгортання лендінгів дозволяє зосередитися на розробці вмісту та функціональності, мінімізуючи витрати та зусилля, пов'язані з побудовою та підтримкою власної інфраструктури, проте важливо також враховувати, що вартість розгортання в хмарних середовищах може залежати від використання ресурсів.

Зваживши всі переваги та недоліки був обран варіант розгортання лендингів в хмарних середовищах. Основними перевагами цього підходу є швидкість розгортання лендінгів, відсутність потреби у побудові та підтримці власної інфраструктури, гнучкість масштабування та висока доступність.

1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

Для успішної реалізації дипломного проєкту будуть використовуватись різноманітні допоміжні програмні засоби, які сприятимуть ефективному функціонуванню системи та спрощенню процесу розробки. Нижче наведено опис певних інструментів, які планується використати у проєкті:

React – це відкрита JavaScript бібліотека для розробки користувацького інтерфейсу. Вона використовується для побудови компонентного підходу до створення веб-додатків, де кожен елемент інтерфейсу є незалежним компонентом зі своїм станом та поведінкою. Завдяки своїй ефективності та простоті використання, React широко використовується у веб-розробці.

Основні переваги React:

– React дозволяє будувати інтерфейс з використанням незалежних компонентів, що спрощує розробку, тестування та підтримку коду. Компоненти можуть бути повторно використані, що забезпечує ефективну розробку та підтримку проекту.

– React використовує віртуальний DOM для оптимізації роботи з реальним DOM. Віртуальний DOM забезпечує швидку перерисовку

елементів тільки у випадках, коли змінюється їх стан. Це робить роботу з інтерфейсом швидшою та ефективнішою.

– React використовує одностороннє зв'язування даних, що означає, що зміни відбуваються тільки в одному напрямку - від батьківського компонента до дочірніх. Це дозволяє забезпечити більш передбачуваний та простий управління станом додатка.

– React дозволяє розширювати функціональність за допомогою сторонніх бібліотек та плагінів. Також існує велика спільнота розробників, що активно підтримує та розробляє розширення для React.

– React може використовуватись для розробки як невеликих веб-додатків, так і великих масштабних проектів. Він добре поєднується з іншими бібліотеками та фреймворками, що дозволяє розширити його можливості[7].

Загалом, використання React для даного проєкту підходить ідеально, тому що цей фреймворк дозволить створити потужний, ефективний та гнучкий інтерфейс для веб-сервісу.

NestJS - це серверний фреймворк для розробки веб-додатків на базі Node.js. Він використовує мову програмування TypeScript, що дозволяє писати код з використанням сучасного синтаксису та переваг статичної типізації. NestJS пропонує архітектурні принципи та шаблони, що базуються на відомих рішеннях, таких як Angular, що спрощує розробку і підтримку додатків.

Переваги NestJS:

– використання TypeScript дозволяє створювати код безпечнішим та більш передбачуваним, завдяки використанню статичної типізації та підтримці сучасного синтаксису.

– NestJS пропонує модульну структуру, що дозволяє організувати проєкт на окремі логічні модулі, спрощуючи розробку, тестування та масштабування.

– NestJS використовує підхід Dependency Injection, що базується на принципах інверсії керування та управління залежностями. Цей підхід дозволяє легко керувати залежностями між компонентами системи та замінювати ці компоненти без зміни коду, що дозволяє спрощувати тестування та розширення функціональності.

– NestJS має розгалужену екосистему, яка включає в себе багато плагінів, модулів та інтеграцій з відомими інструментами, такими як Swagger та інші. Ця міцна екосистема дозволяє розширювати функціональність NestJS і використовувати його в поєднанні з іншими потужними інструментами для розробки веб-додатків[8].

Зважаючи на це, NestJS був обран для написання серверної частини проєкту, що дозволить писати безпечний, структурований та легко розширюваний код.

i18next - це бібліотека JavaScript для локалізації веб-додатків. Вона надає зручні інструменти для керування текстовими ресурсами, такими як переклади, мовні ключі, форматування дат і чисел, що дозволяє легко локалізувати додаток на різні мови.

Основні особливості і переваги i18next:

– i18next дозволяє легко розміщувати всі текстові ресурси додатку у відповідних мовних файлах. Це дозволяє швидко перекладати додаток на різні мови і підтримувати багатомовність.

– i18next дозволяє використовувати мовні ключі для ідентифікації текстових ресурсів. Це забезпечує зручну організацію і керування перекладами в мовних файлах.

– i18next підтримує різні формати мовних файлів, такі як JSON, YAML, Gettext, XML та інші. Це дозволяє працювати з різними інструментами локалізації та імпортувати/експортувати переклади з різних джерел[9].

React Router - це бібліотека JavaScript, яка надає маршрутизацію для веб-додатків, написаних з використанням фреймворка React. Вона дозволяє

легко управляти навігацією та переходами між різними сторінками або компонентами в додатку.

Основні особливості React Router:

– React Router використовує декларативний підхід до визначення маршрутів. Можна використовувати компоненти, такі як <Route> та <Link>, для опису шляхів та навігації, що робить код більш зрозумілим.

– React Router дозволяє визначати динамічні маршрути, які залежать від параметрів URL. Можна передавати параметри в URL та легко отримувати доступ до них в компонентах додатку.

– React Router підтримує вкладену маршрутизацію, що означає, що можна вкладати маршрути один в одного. Це дозволяє створювати складну структуру навігації та управляти різними рівнями вкладеності в додатку.

– React Router дозволяє легко управляти історією переходів між сторінками. Можна використовувати методи, такі як history.push() або history.replace(), для навігації між сторінками або заміни поточного стану.

– React Router підтримує різні типи маршрутів, включаючи статичні маршрути, параметризовані маршрути, маршрути з опціональними параметрами та багато інших. Це дозволяє гнучко визначати правила навігації в додатку[10].

Загалом, React Router є потужним інструментом для керування маршрутизацією в React-додатках, дозволяючи створювати динамічну та зручну навігацію між різними сторінками або компонентами додатку.

HTML є стандартною мовою розмітки для веб-сторінок. Вона дозволяє організовувати та структурувати вміст за допомогою тегів, таких як заголовки, параграфи, зображення та посилання. Разом з CSS та JavaScript, HTML допомагає створювати привабливі, інформативні та інтерактивні веб-додатки. Вона також надає можливості для визначення семантики та доступності веб-сторінок, сприяючи кращому розумінню контенту пошуковим системам та іншим програмам. HTML є необхідним інструментом для будь-якого веб-розробника і використовується для

створення коректно структурованих та сумісних з різними браузерами веб-сторінок.

SCSS – це скриптова метамова, яка інтерпретується в каскадні таблиці стилів (CSS), що означає Sassy CSS, є розширеною версією CSS і препроцесором, який надає додаткові функції та можливості для покращення розробки стилів у веб-проектах. За допомогою SCSS, розробники можуть писати CSS код у більш організованому, зручному та ефективному форматі, який потім компілюється до звичайного CSS, зрозумілого браузерам.

Однією з ключових переваг SCSS є можливість використання змінних, які дозволяють задавати значення, які використовуються повторно у стилях. Це робить зміну стилів на всій веб-сторінці більш простою та зручною. Вкладені селектори дозволяють вкладати один селектор всередину іншого, що полегшує організацію та структурування CSS правил. Крім того, міксіни (mixins) дозволяють створювати блоки стилів, які можуть бути повторно використані в різних частинах проекту.

PostgreSQL є потужною та розширеною об'єктно-реляційною системою керування базами даних (ORDBMS). Вона є однією з найпопулярніших відкритих систем у світі та здатна виконувати різноманітні завдання від невеликих проектів до великих підприємственних додатків.

Основні особливості PostgreSQL включають підтримку SQL запитів, транзакційної безпеки, використання зовнішніх ключів, розширені можливості індексування та оптимізації запитів. Вона також підтримує різні типи даних, включаючи числа, рядки, дати, час, JSON, XML і багато інших. PostgreSQL пропонує багато вбудованих функцій, таких як агрегатні функції, функції для обробки тексту, географічні функції та інші, що полегшують роботу з даними[11].

Переваги PostgreSQL полягають у його надійності, масштабованості та розширюваності. Вона підтримує реплікацію для забезпечення високої доступності та забезпечення надійності даних.

Redis є відкритою, високошвидкісною та ключ-значення (key-value) системою зберігання даних. Вона використовується для зберігання та управління даними в оперативній пам'яті, що дозволяє надзвичайно швидко отримувати доступ до даних.

Redis надає розширені функції, які дозволяють використовувати його в різних сценаріях, включаючи кешування, сесії, повідомлення, розподілені блокування та багато іншого. Деякі з його основних переваг включають:

– швидкодію, так як Redis працює в оперативній пам'яті та використовує оптимізовані алгоритми зберігання та доступу до даних, що робить його дуже швидким. Він може обробляти мільйони операцій на секунду, що робить його ідеальним для вимогливих застосунків, які потребують низької латентності.

– Redis є гнучким, оскільки він підтримує різні типи даних, такі як рядки, хеші, списки, множини та сортовані множини. Це означає, що можна моделювати різні типи даних і використовувати Redis для різних цілей, таких як кешування, індексування та аналітика. Завдяки такій гнучкості Redis може використовуватися в різних сценаріях і відповідати на потреби різних проектів.

– Redis забезпечує можливість зберігання даних на диску, що дозволяє їм залишатися доступними й після перезапуску сервера. Це призводить до стійкості Redis до відмов та забезпечує постійний доступ до даних. Це особливо важливо в ситуаціях, коли сервер перезавантажується або виникає відмова, оскільки це дозволяє зберегти дані та забезпечити безперервний доступ до них.

– Redis надає розширені можливості, оскільки має великий набір команд для роботи з даними. Ці команди дозволяють виконувати різноманітні операції з даними, такі як сортування, фільтрація, обчислення наборів даних та інші. Більше того, Redis підтримує транзакції, які дозволяють виконувати групу операцій над наборами даних атомарно, тобто забезпечувати їх виконання в цілісному стані[12].

1.3.3 Аналіз відомих програмних продуктів

WordPress – це одна з найпопулярніших платформ для створення веб-сайтів і блогів. Воня має широкий спектр тем та плагінів, що дозволяють генерувати і налаштовувати лендинги. Є також спеціалізовані теми та плагіни, призначені саме для створення лендинг-сторінок. WordPress надає зручний інтерфейс для редагування вмісту, налаштування макетів, інтеграцію з різними платформами і розширенням функціональності за допомогою плагінів. Для розгортання лендингів на WordPress потрібно мати веб-хостинг і налаштувати цю платформу на своєму сервері[13].

Переваги:

– широкий вибір тем та плагінів;

– гнучкість та налаштування;

– розширення функціональності;

– зручний інтерфейс редагування вмісту;

– спільнота та підтримка;

Недоліки:

– складність для новачків;

– постійні оновлення;

– при використанні багатьох плагінів та налаштувань в WordPress можуть виникати проблеми з продуктивністю та швидкодією веб-сайту.

Wix – це хмарна платформа для створення веб-сайтів, яка надає інструменти для створення професійних лендингів. Вона має вбудований конструктор, який дозволяє легко створювати та налаштовувати лендинги за допомогою перетягування та розміщення елементів. Wix пропонує багато готових шаблонів, які можна використовувати як основу для лендингів. Вона також надає зручні інструменти для оптимізації для пошукових систем (SEO), аналітики та моніторингу відвідуваності. Розгортання лендингів на Wix відбувається на серверах цієї платформи[14].

Переваги:

– багатий вибір дизайнів;

– функціональність;

– гнучкість;

– хостинг та безпека;

Недоліки:

– вартість;

– може бути складним у використанні для новачків або користувачів з обмеженим досвідом у роботі з платформами для створення лендінгів.

– власність контенту;

– залежність від хостингу;

Unbounce – це спеціалізована платформа для створення лендинг-сторінок, яка дозволяє легко створювати, тестувати та оптимізувати лендинги для маркетингових кампаній. Unbounce надає широкий вибір готових шаблонів та інструментів для персоналізації лендингів. Його конструктор має простий та інтуїтивно зрозумілий інтерфейс, що дозволяє редагувати вміст, стилі та макети лендингів. Unbounce також надає можливості A/B-тестування для визначення найефективніших варіантів лендингів. Розгортання лендингів на Unbounce відбувається на серверах цієї платформи[15].

Переваги:

– гнучкість та налаштування;

– можливості A/B-тестування;

– висока швидкодія;

– служба підтримки;

Недоліки:

– вартість;

– обмежена свобода дизайну;

– відсутність інтеграцій;

Порівняння веб-сервісу для генерації та розгортання лендингів, під назваю LandingBoost з аналогами в таблиці 1.1.

Таблиця 1.1 – Порівняння з аналогом

Функціонал	LandingBoost	WordPress	Wix	Unbounce	Пояснення
Зручний та простий інтерфейс створення та редагування веб-сторінок	+	+	-	+	Хмарна платформа Wix може бути складною для новачків.
Створення веб-сторінки з існуючого шаблона	+	+	+	+	Кожен програмний продукт надає користовачу можливість створити веб-сторінку з шаблону.
Редагування вмісту веб-сторінок	+	+	+	+	Всі програмні продукти мають можливість редагування вмісту веб-сторінок.
Редагування дизайну веб-сторінок	+	+	+	+	Кожна платформа надає можливості для редагування дизайну веб-сторінок.

Продовження таблиці 1.1

Функціонал	LandingBoost	WordPress	Wix	Unbounce	Пояснення
Редагування HTML/CSS/ JavaScript коду	-	+	-	+	Веб-сервіс для генерації та розгортання лендингів та платформа Wix не надає можливість користувачу редагувати безпосередньо HTML/CSS/ JavaScript код.
Аналітика та звіти про продуктивність	-	-	+	+	Веб-сервіс для генерації та розгортання лендингів не надає аналітику та звіти про продуктивність, а в WordPress це залежить від плагінів.
Безкоштовне створення веб-сторінок	+	+	-	-	Програмні продукти Wix та Unbounce є вартісними платформами, з високими цінами.

1.4 Аналіз вимог до програмного забезпечення

Головною функцією програмного забезпечення є генерація лендингів, більше функцій можна побачити на діаграмі варіантів використання, яка наведена у графічних матеріалах «Схема структурна варіантів використання».

В таблицях 1.2 - 1.8 наведені варіанти використання програмного забезпечення.

Таблиця 1.2 - Варіант використання UC-1

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Реєстрація нового користувача в системі.
Actors	Незареєстрований користувач.
Trigger	Користувач заповнює форму реєстрації та натискає кнопку «Зареєструватися».
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку реєстрації. В поля для реєстрації вводяться відповідні особисті дані користувача, такі як: ім'я, прізвище, адресу електронної пошти та пароль. Після заповнення даних, користувач натискає кнопку «Зареєструватися». Після цього система створює новий обліковий запис та перенаправляє користувача на його сторінку.
Extension	В випадку введення не коректних даних, кнопка реєстрації стає неактивною. Якщо якесь конкретне поле введено некоректно, то воно підсвічується червоним надписом.
Post-Condition	Створення сторінки користувача, перехід на сторінку користувача.

Таблиця 1.3 - Варіант використання UC-2

Use case name	Аутентифікація користувача.
Use case ID	UC-02
Goals	Підтвердження ідентифікації користувача.
Actors	Користувач
Trigger	Користувач відкриває сторінку аутентифікації та бажає увійти в свій обліковий запис.
Pre-conditions	Користувач повинен мати обліковий запис на платформі.
Flow of Events	Користувач переходить на сторінку аутентифікації та вводить свої адресу електронної пошти та пароль. Система перевіряє, чи існує обліковий запис з такою електронною поштою та паролем. Якщо обліковий запис існує, то система надає доступ користувачу до веб-сервісу.
Extension	Якщо обліковий запис не знайдено, система повідомляє про невірний email або пароль та пропонує спробувати ще раз.
Post-Condition	Користувач успішно авторизувався в системі та має доступ до функціоналу веб-сервісу.

Таблиця 1.4 - Варіант використання UC-3

Use case name	Створення веб-сторінки.
Use case ID	UC-03
Goals	Користувач може створити веб-сторінку.
Actors	Користувач
Trigger	Користувач авторизується.
Pre-conditions	Користувач має акаунт на платформі.
Flow of Events	На домашній сторінці користувач натискає кнопку «+». Платформа показує усі доступні шаблони веб-сторінок. Користувач обирає шаблон, що сподобався.
Extension	-

Продовження таблиці 1.4

Post-Condition	Нова веб-сторінка з відповідним шаблоном створена та збережена в системі.

Таблиця 1.5 - Варіант використання UC-4

Use case name	Перегляд списку створених веб-сторінок.
Use case ID	UC-04
Goals	Користувач може переглянути список створених веб-сторінок.
Actors	Користувач
Trigger	Користувач авторизується.
Pre-conditions	Користувач має акаунт на платформі та створив хоча б одну веб-сторінку.
Flow of Events	На домашній сторінці користувач бачить всі створені веб-сторінки. Платформа показує назву, короткий опис та дату створення веб-сторінки.
Extension	-
Post-Condition	Користувач може перейти в режим редагування веб-сторінки.

Таблиця 1.6 - Варіант використання UC-5

Use case name	Редагування дизайну веб-сторінки.
Use case ID	UC-05
Goals	Користувач може редагувати дизайн веб-сторінки.
Actors	Користувач
Trigger	Користувач авторизується та заходить в режим редагування веб-сторінки.
Pre-conditions	Користувач має акаунт на платформі та створив хоча б одну веб-сторінку.

Продовження таблиці 1.6

Flow of Events	На домашній сторінці користувач вибирає потрібну для редагування веб-сторінку та заходить в режим редагування. В режимі редагування користувач змінює дизайн веб-сторінки по своїм потребам та настискає кнопку «Зберегти».
Extension	-
Post-Condition	Користувач успішно змінив дизайн веб-сторінки та зберіг зміни.

Таблиця 1.7 - Варіант використання UC-6

Use case name	Редагування вмісту веб-сторінки.
Use case ID	UC-06
Goals	Користувач може редагувати вміст веб-сторінки.
Actors	Користувач
Trigger	Користувач авторизується та заходить в режим редагування веб-сторінки.
Pre-conditions	Користувач має акаунт на платформі та створив хоча б одну веб-сторінку.
Flow of Events	На домашній сторінці користувач вибирає потрібну для редагування веб-сторінку та заходить в режим редагування. В режимі редагування користувач змінює вміст веб-сторінки по своїм потребам та настискає кнопку «Зберегти».
Extension	-
Post-Condition	Користувач успішно змінив вміст веб-сторінки та зберіг зміни.

Таблиця 1.8 - Варіант використання UC-7

Use case name	Збереження веб-сторінки.
Use case ID	UC-07
Goals	Користувач зберігає веб-сторінку.
Actors	Користувач

Продовження таблиці 1.8

Trigger	Користувач авторизується та заходить в режим редагування веб-сторінки.
Pre-conditions	Користувач має акаунт на платформі та створив хоча б одну веб-сторінку.
Flow of Events	На домашній сторінці користувач вибирає потрібну для редагування веб-сторінку та заходить в режим редагування. В режимі редагування користувач вносить будь-які зміни в веб-сторінку та настискає кнопку «Зберегти».
Extension	-
Post-Condition	Користувач успішно зберіг зміни на веб-сторінці.

1.4.1 Розроблення функціональних вимог

Програмне забезпечення поділено на окремі модулі, кожен з яких виконує свою функцію. Детальний опис функціональних вимог можна знайти в таблицях 1.9-1.13, а на рисунку 1.1 наведено загальну модель вимог. Матриця трасування вимог на рисунку 1.2 відображає зв'язок між вимогами і модулями програми.

Вимога	Код	Пріоритет	Ризик
1. Реєстрація користувача	FR-1	Високий	Середній
1.1 Валідація полів	FR-1.1	Середній	Низький
1.3 Авторизація користувача	FR-1.2	Високий	Середній
1.4 Валідація полів	FR-1.3	Середній	Низький
2. Створення веб-сторінки	FR-2	Високий	Високий
3. Перегляд створених веб-сторінок	FR-3	Середній	Високий
4. Редагування веб-сторінок	FR-4	Високий	Середній
5. Збереження веб-сторінок	FR-5	Високий	Високий

Рисунок 1.1 – Модель вимог у загальному вигляді

Таблиця 1.9 – Функціональна вимога FR-1

Назва	Авторизації користувача
Опис	Система повинна надавати можливість реєстрації користувачеві шляхом введення ім'я, прізвища, адресу електронної пошти та паролю, та авторизації користувача шляхом введення адресу електронної пошти та паролю.

Таблиця 1.10 – Функціональна вимога FR-2

Назва	Створення веб-сторінки користувачем
Опис	Користувачі можуть створювати веб-сторінки в своєму акаунті.

Таблиця 1.11 – Функціональна вимога FR-3

Назва	Перегляд вже створенних веб-сторінок
Опис	Система надає можливість користувачу переглядати вже створені веб-сторінки, та коротку інформацію про них.

Таблиця 1.12 – Функціональна вимога FR-4

Назва	Редагування веб-сторінок
Опис	Користувач може редагувати веб-сторінки в залежності від своїх потреб.

Таблиця 1.13 – Функціональна вимога FR-5

Назва	Збереження веб-сторінок
Опис	Після редагування шаблону користувач зберігає зміни для своєї веб-сторінки.

	FR-1	FR-2	FR-3	FR-4	FR-5
UC-1	+				
UC-2	+				
UC-3		+			
UC-4			+		
UC-5				+	
UC-6				+	
UC-7					+

Рисунок 1.2 – Матриця трасування вимог

1.4.2 Розроблення нефункціональних вимог

Нижче приведені вимоги до програмного забезпечення, які не стосуються його функціональності:

– забезпечення високого рівня безпеки є невід'ємною частиною розробки програмного забезпечення. Для досягнення цього використовуються різні підходи до аутентифікації та авторизації, шифрування даних, захисту від атак та зловживань, а також забезпечення конфіденційності, цілісності та доступності даних.

– програмне забезпечення має забезпечувати оптимальну продуктивність та швидкість роботи, навіть при великому обсязі робочих навантажень та обробці великих обсягів даних. При цьому важливо дотримуватись встановленого ліміту максимального часу відповіді, щоб забезпечити безперебійну та плавну функціональність системи.

– програмне забезпечення повинно мати високий рівень надійності та стійкості до відмов. Це досягається шляхом проведення тестування для виявлення помилок та вразливостей, наявності резервного копіювання даних, механізмів відновлення та забезпечення безперебійної роботи системи.

– програмне забезпечення повинно бути сумісним з різними операційними системами, пристроями та браузерами, щоб забезпечити його взаємодію в різних середовищах. Це означає, що програмне забезпечення повинно використовувати відповідні інтерфейси, стандарти та протоколи, що дозволять йому працювати безпроблемно з різними системами та пристроями.

1.5 Постановка задачі

Задача полягає в розробці функціональності та забезпеченні потрібних можливостей користувачам, які хочуть створити привабливі та ефективні веб-сторінки для своїх продуктів чи послуг.

Основні етапи постановки задачі:

– аналіз потреб користувачів: Визначення вимог та очікувань користувачів щодо функціональності сервісу. Розуміння їх потреб у генерації та розгортанні лендингів, а також їх вимог щодо дизайну, налаштувань.

– визначення основних функціональних можливостей: Встановлення ключових функцій, які повинні бути реалізовані в сервісі, таких як вибір шаблону, редагування тексту та дизайну, додавання елементів та розгортання лендингів.

– розробка інтерфейсу користувача: Створення зручного та інтуїтивно зрозумілого інтерфейсу, який дозволить користувачам легко використовувати функціонал сервісу.

– розробка алгоритмів та логіки роботи: Реалізація алгоритмів, які дозволять здійснювати генерацію лендингів на основі обраних шаблонів, забезпечувати можливість редагування та налаштування вмісту, а також збереження та розгортання створених лендингів.

– забезпечення безпеки та надійності: Реалізація заходів безпеки, щоб захистити дані користувачів, забезпечити автентифікацію та авторизацію, а

також забезпечити резервне копіювання та можливість відновлення даних в разі відмови.

– тестування та вдосконалення: Проведення ретельного тестування всіх функціональних можливостей, виявлення та виправлення помилок та недоліків. Постійне вдосконалення сервісу.

Метою проєкту є спрощення та прискорення процесу створення та розгортання лендингів з метою вирішення проблеми складності і затрат, пов'язаних з цими процесами. Це дозволить користувачам ефективно просувати свої продукти та послуги, привертати увагу клієнтів та забезпечувати успішність їх бізнесу.

Висновки до розділу

У даному розділі було проведено аналіз вимог до програмного забезпечення, спрямованого на розробку веб-сервісу для генерації та розгортання лендингів. З метою розуміння потреб користувачів та успішної реалізації проєкту були визначені ключові цілі. Також було розглянуто основні функціональні вимоги до веб-сервісу. Було описано різноманітні можливості платформи, такі як генерація веб-сторінок та їх розгортання, які сприяють ефективному просуванню продуктів та послуг.

Крім того, були надані детальні описи для кожної функціональної вимоги, що дозволяють краще зрозуміти вимоги користувачів та специфіку їх використання. Це допомагає створити зручний та ефективний інтерфейс для взаємодії з сервісом.

Окрім того, були визначені додаткові вимоги до інтерфейсу користувача, забезпечення безпеки даних та приватності користувачів. Ці вимоги покликані забезпечити зручність та надійність використання веб-сервісу.

У результаті аналізу було встановлено, що успішна реалізація проєкту вимагатиме розробки потужної системи для генерації веб-сторінок та їх

автоматичного розгортання. Потрібно ретельно розробити архітектуру та функціональність, забезпечити зручний інтерфейс користувача та високу ступінь безпеки.

Загальною метою проєкту є надання ефективного інструменту для розробки лендінгів, що відповідає вимогам та потребам користувачів. Аналіз вимог до програмного забезпечення є першим кроком у цьому напрямку, допомагаючи зрозуміти ключові потреби та визначити напрямки роботи над проєктом.

Наступним етапом буде моделювання та конструювання програмного забезпечення, який враховуватиме встановлені вимоги та дозволять реалізувати проєкт згідно з поставленими цілями.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

У цьому розділі буде проведено моделювання та аналіз програмного забезпечення проєкту "Веб-сервіс для генерації та розгортання лендингів". Моделювання програмного забезпечення є важливим етапом у розробці проєкту, оскільки дозволяє зрозуміти його структуру, взаємозв'язки між компонентами та поведінку системи.

Основний бізнес процес, який включає взаємодію користувача та веб-сервісу у контексті створення, редагування та збереження веб-сторінки, може бути описаний таким чином:

– користувач переглядає існуючі шаблони. Користувач авторизувавшись, має можливість переглядати та вибирати існуючі шаблони.

– користувач створює веб-сторінку. Після перегляду та вибору підходящого шаблону, користувач створює власну веб-сторінку на основі обраного шаблону.

– користувач редагує веб-сторінку. Створивши веб-сторінку, користувач отримує можливість редагувати відповідні текстові поля, обирати кольорову гаму з доступних для цього шаблону в залежності від своїх потреб.

– користувач зберігає веб-сторінку. Відредагувавши веб-сторінку, користувач може зберегти веб-сторінку з усіма змінами. Потім користувач може дивитися список усіх своїх створених веб-сторінок з коротким описом та датою створення веб-сторінки.

Цей процес забезпечує користувачам, навіть тим, хто має обмежений досвід у роботі з платформами для створення лендінгів, можливість легко та швидко створити свій власний лендінг. Це досягається завдяки простому та зрозумілому інтерфейсу. Навіть новачки можуть легко орієнтуватися та створювати привабливі та функціональні лендінги без особливих зусиль.

Для опису бізнес процесу програмного забезпечення використовується BPMN модель (рисунок 2.1).

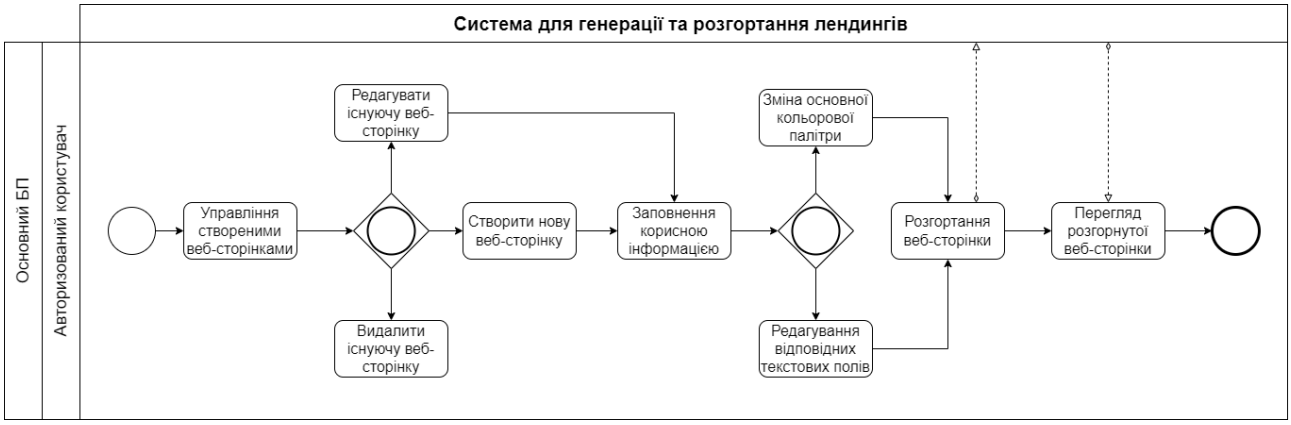


Рисунок 2.1 – BPMN схема бізнес-процесу

2.2 Архітектура програмного забезпечення

В проєкті "Веб-сервіс для генерації та розгортання лендингів" реалізована монолітна архітектура, це означає, що увесь код та функціонал розміщено в одній програмі (моноліті). Фреймворк NestJS використовується для реалізації цієї архітектури.

У монолітній архітектурі платформи використовується тришарова модель, така як MVC (рисунок 2.2). Ця модель розділяє систему на три компоненти, які взаємодіють між собою.

Модель (Model). Вона відповідає за управління даними, які використовуються в системі та бізнес-логікою. Модель обробляє запити, проводить необхідні операції з базою даних та забезпечує цілісність та доступ до даних, пов'язаних із управлінням, збереженням, оновленням, отриманням даних та розгортанням лендингів.

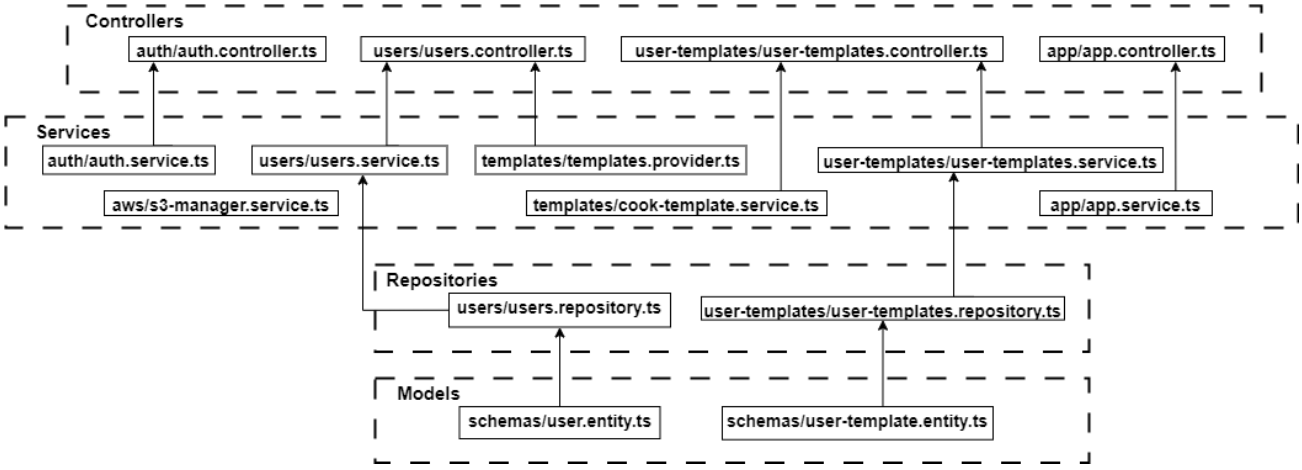


Рисунок 2.2– Шари архітектури програмного забезпечення

Представлення (View). Воно відповідає за відображення даних та взаємодію з користувачем. Представлення надає користувачеві можливість взаємодіяти з веб-сервісом, вводити дані та переглядати результати. У контексті проєкту "Веб-сервіс для генерації та розгортання лендингів" представлення може мати такі складові. Інтерактивний інтерфейс, який дозволяє користувачам створювати та редагувати свої лендинги. Він може включати можливості для додавання тексту, зображень та інших елементів, необхідних для створення ефективного лендінгу. Редактор лендингів має бути інтуїтивно зрозумілим, навіть для користувачів з обмеженим досвідом у роботі зі схожими інструментами. Крім редактора лендингів, представлення також може включати інтерфейс, який дозволяє користувачам налаштовувати різні параметри свого лендінгу. Це можуть бути параметри, пов'язані з дизайном, кольорами та іншими аспектами, які дозволяють користувачам персоналізувати свій лендінг та адаптувати його під свої потреби. Важливо, щоб представлення було зрозумілим, зручним та інтуїтивно зрозумілим для користувачів, незалежно від їхнього рівня досвіду в роботі з подібними інтерфейсами. Представлення також забезпечує взаємодію з користувачем шляхом обробки його введених даних, запитів та відображення результатів. Воно може включати форми для введення даних, кнопки для виконання певних дій та інші елементи, які дозволяють користувачеві ефективно спілкуватися з веб-сервісом.

Контролер (Controller). В даному проєкті, контролер відіграє важливу роль у взаємодії з іншими компонентами системи. Його основними завданнями є обробка запитів, маршрутизація та ініціювання необхідних дій. Контролер отримує запити від користувача через представлення, які можуть бути введенням даних, натисканням кнопок або іншими діями, здійсненими в інтерфейсі. Контролер аналізує ці запити та визначає, які дії потрібно виконати для їх обробки. Наприклад, якщо користувач натискає кнопку "Зберегти", контролер ініціює процес збереження даних лендінгу. Після отримання запиту контролер взаємодіє з іншими компонентами системи для виконання необхідних дій. Наприклад, він може взаємодіяти з моделлю, щоб отримати або зберегти дані лендінгу в базі даних. Контролер також може спілкуватися з іншими службами або модулями системи для виконання специфічних функцій, наприклад, для збереження зображень у хмарному сховищі або обробки даних. Після виконання необхідних дій контролер повертає відповідь до представлення. Ця відповідь може бути результатом виконання запиту, повідомленням про стан операції або іншою інформацією, яка повинна бути відображена користувачеві. Представлення отримує цю відповідь та відображає її користувачу, що дозволяє забезпечити взаємодію з веб-сервісом та надати необхідну інформацію користувачеві. Таким чином, контролер виступає як посередник між представленням та іншими компонентами системи, забезпечуючи обробку запитів, маршрутизацію та взаємодію з іншими компонентами для досягнення функціональності проєкту[16].

У проєкті "Веб-сервіс для генерації та розгортання лендингів" в якості баз даних використовується PostgreSQL та Redis. Кожна з цих баз даних виконує свої функції та має свою доцільність у контексті проєкту.

PostgreSQL є потужною та надійною об'єктно-реляційною базою даних, яка використовуєся для зберігання структурованих даних. У проєкті, PostgreSQL використовується для зберігання і керування різноманітною інформацією, пов'язаною з лендінгами. Наприклад, зберігає дані про

користувачів, їхні лендінги, параметри налаштування та інші важливі дані. PostgreSQL надає широкі можливості для роботи зі структурованими даними та підтримує розширення SQL-запитів, що дозволяє ефективно здійснювати операції з базою даних.

Redis є швидким та потужним інструментом для зберігання та кешування даних. Платформа може використовувати Redis для зберігання та управління тимчасовими даними, швидкого доступу до певних ресурсів та кешування результатів запитів. Наприклад, Redis може використовуватись для кешування сторінок лендінгу, що дозволить швидко відображати їх користувачам без необхідності повторного генерування. Крім того, Redis має розширені можливості для роботи зі структурованими даними, списками, хешами та іншими типами даних, що дозволяє ефективно виконувати операції над ними.

Загальною доцільністю використання PostgreSQL і Redis у проєкті є забезпечення потужних та ефективних механізмів зберігання, керування та доступу до даних. Наприклад, зберігає дані про користувачів, їхні лендінги, параметри налаштування та інші важливі дані. PostgreSQL забезпечує стабільну та надійну роботу зі структурованими даними, а Redis дозволяє швидко доступатись до тимчасових даних та кешувати результати для покращення продуктивності веб-сервісу.

Клієнтська частина веб-сервісу написана за допомогою React з використанням компонентного підходу.

У React компоненти використовуються для створення візуальних елементів та логіки взаємодії з користувачем. Кожен компонент представляє собою окрему частину користувацького інтерфейсу, яка може бути повторно використана та має свою внутрішню логіку та стан.

Доцільність використання React та компонентного підходу для проєкту "Веб-сервіс для генерації та розгортання лендингів" полягає в наступних перевагах:

– модульність. За допомогою компонентного підходу можна розділити інтерфейс на невеликі незалежні компоненти, що спрощує розробку, тестування та підтримку коду. Кожен компонент може бути розроблений, протестований та модифікований окремо, що полегшує роботу над проєктом та сприяє його розширенню.

– компонентний підход. Завдяки модульному підходу, можна створювати компоненти, які можуть бути використані на різних сторінках веб-сервісу або навіть в інших проектах. Це забезпечує ефективність розробки, оскільки компоненти можуть бути перевикористані замість того, щоб писати код з нуля.

– декларативний підхід. React пропонує декларативний підхід до опису інтерфейсу, де розробник зосереджується на тому, що повинно бути відображено, а не як саме це має бути досягнуто. Це спрощує розробку та зрозуміння коду, а також дозволяє швидко вносити зміни та модифікувати інтерфейс.

– спільнота розробників. React має велику спільноту розробників, що сприяє доступності документації, бібліотек, компонентів та інструментів. Це полегшує використання сторонніх рішень, розширює можливості розробки та прискорює процес розробки веб-сервісу.

Клієнтська частина, написана з використанням React та компонентного підходу, дозволяє забезпечити ефективну розробку та підтримку веб-сервісу для генерації та розгортання лендингів. Вона надає модульність, повторне використання та декларативність.

2.3 Конструювання програмного забезпечення

Під час розробки проєкту "Веб-сервіс для генерації та розгортання лендингів" було виконано конструювання програмного забезпечення, що включало розробку архітектури, алгоритмів, структур даних та інших

компонентів системи, основні класи наведені у графічних матеріалах «Схема структурна класів програмного забезпечення».

В процесі конструювання була розроблена архітектура програмного забезпечення, що визначала організацію компонентів та їх взаємодію. Для проєкту була обрана монолітна архітектурна парадигма, що дозволяє забезпечити простоту розгортання та масштабування системи.

Протягом розробки платформи були створені оригінальні алгоритми, які мають значну вагу в роботі системи. Опис цих алгоритмів наведено нижче для більш детального розуміння:

– алгоритм генерації лендингів. Цей алгоритм використовується для створення лендингів для кожного користувача. Він базується на комбінації різних параметрів, таких як шаблон лендингу, вміст, кольорова схема тощо. Алгоритм генерує унікальний ідентифікатор для кожного лендингу, що забезпечує їхню унікальність та відрізняє їх один від одного.

– алгоритм обробки користувацьких запитів. Цей алгоритм відповідає за обробку запитів, що надходять від користувачів системи. Він аналізує отримані дані, перевіряє їхню коректність та виконує необхідні дії згідно з логікою системи. Наприклад, він може обробляти запити на створення нового лендингу, зміну налаштувань тощо.

– алгоритм маршрутизації. Цей алгоритм визначає, які запити повинні бути спрямовані до яких компонентів системи. Він враховує шляхи до різних функціональних модулів, базуючись на отриманому запиті та його характеристиках. Алгоритм маршрутизації гарантує, що кожен запит буде відправлений до відповідного компонента для подальшої обробки.

Ці алгоритми були ретельно розроблені та оптимізовані з метою забезпечення високої продуктивності та ефективної роботи системи. Вони відіграють ключову роль у забезпеченні швидкої та надійної генерації лендингів, обробці користувацьких запитів та правильному маршрутизації даних у системі.

Було створено структури даних, що дозволяють ефективно зберігати та обробляти інформацію. Наприклад, структури для зберігання шаблонів лендингів, користувацьких даних. Використання оптимальних структур даних сприяє швидкій обробці запитів та ефективному використанню ресурсів системи.

Також у процесі конструювання були використані різні сторонні утиліти, бібліотеки та фреймворки, які допомогли спростити розробку та забезпечити необхідні функціональні можливості. Наприклад, використання фреймворку React дозволило реалізувати компонентний підхід та забезпечити зручний інтерфейс для користувачів. Використання фреймворку NestJS було важливим для реалізації архітектури та взаємодії між компонентами моделі, представлення та контролера. NestJS надавав структуру та організацію коду, спрощуючи розробку та забезпечуючи ефективну маршрутизацію запитів. Так само, в процесі конструювання, була використана бібліотека Axios для здійснення HTTP-запитів. Використання Axios у проєкті дозволило спростити взаємодію з сервером і здійснювати різноманітні запити, такі як отримання, відправлення або оновлення даних. Бібліотека підтримує різні методи запитів, такі як GET, POST, PUT, DELETE[17]. Використання Axios дозволило ефективно взаємодіяти з бекендом системи, обмінюватись даними і забезпечувати необхідну функціональність веб-сервісу. Завдяки широким можливостям налаштування, Axios став важливим інструментом для роботи з HTTP-запитами у проєкті.

Крім того, конструювання програмного забезпечення включало процес тестування та налагодження, що дозволяє перевірити функціональність, коректність та надійність системи.

В результаті конструювання програмного забезпечення для проєкту "Веб-сервіс для генерації та розгортання лендингів" була створена система з оптимальною архітектурою, ефективними алгоритмами та структурами даних. Це дозволяє забезпечити швидку та надійну роботу сервісу, задовольняючи потреби користувачів у генерації та розгортанні лендингів.

В якості системи управління базами даних використовується PostgreSQL і Redis. База даних серверу призначена для зберігання і керування різноманітною інформацією, пов'язаною з лендінгами. Схема бази даних наведена у графічних матеріалах «Схема бази даних», складається з таблиці користувачів зі зв’язком «один до багатьох» по відношенню до таблиці шаблонів користувачів. Опис таблиць бази даних наведено у таблицях 2.1 – 2.2.

Таблиця 2.1 – Опис таблиці users

Таблиця	Назва поля	Тип даних	Опис
users	id	integer	Первинний ключ
	name	varchar	Ім’я користувача
	surname	varchar	Прізвище користувача
	password	varchar	Зашифрований пароль користувача
	email	varchar	Електронна адреса користувача
	verified	boolean	Флаг підтвердження електронної пошти
	Deleted	boolean	Ідентифікація наявності акаунту на платформі
	createdAt	timestamp	Дата створення
	updatedAt	timestamp	Дата оновлення

Таблиця 2.2 – Опис таблиці users_templates

Таблиця	Назва поля	Тип даних	Опис
users_templates	id	integer	Первинний ключ
	userId	integer	Вторинний ключ на таблицю «users»
	data	text	Данні, що стосуються корисного навантаження шаблону у JSON форматі
	createdAt	timestamp	Дата створення
	updatedAt	timestamp	Дата оновлення

Наведений у таблиці 2.3 опис включає утиліти, бібліотеки та інше стороннє програмне забезпечення, яке було використано під час розробки веб-сервісу.

Таблиця 2.3 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Docker Desktop	Docker Desktop забезпечує зручне графічне інтерфейсне середовище, яке дозволяє легко управляти контейнерами, мережами, образами та іншими компонентами Docker.
2	VS Code	Редактор коду, який надає розширені можливості для написання, редагування та налагодження програмного коду різних мов програмування.

Продовження таблиці 2.3

№ п/п	Назва утиліти	Опис застосування
3	PgAdmin	Платформа для адміністрування та розробки PostgreSQL та пов'язаних з ним систем управління базами даних надає засоби для управління, адміністрування та розробки баз даних, які використовують PostgreSQL та його супутні системи управління базами даних.

2.4 Аналіз безпеки даних

У контексті проєкту "Веб-сервіс для генерації та розгортання лендингів" безпека даних є надзвичайно важливим аспектом. Під час аналізу безпеки даних виконується оцінка потенційних загроз, визначення вразливостей, ідентифікація ризиків та розробка заходів для їх запобігання та мінімізації.

Оцінка загроз передбачає аналіз потенційних загроз безпеці даних, які можуть виникнути внаслідок зловживання, несанкціонованого доступу, атак з боку хакерів, втрати або крадіжки даних тощо. Загрози можуть включати такі елементи, як вразливості в системі, недостатню аутентифікацію та авторизацію, атаки на перехоплення даних, атаки на введення шкідливого коду тощо.

Визначення вразливостей пов'язане з ідентифікацією слабких місць у системі, які можуть бути використані для порушення безпеки даних. Це можуть бути проблеми у налаштуваннях сервера, вразливість у програмному забезпеченні, недостатній рівень шифрування даних тощо.

Ідентифікація ризиків включає оцінку потенційних наслідків, які можуть виникнути внаслідок вразливостей і загроз безпеці даних. Ризики

можуть включати втрату даних, порушення конфіденційності, недоступність системи, негативний вплив на репутацію компанії тощо.

Розробка заходів для запобігання та мінімізації ризиків передбачає прийняття заходів безпеки, що допомагають запобігти атакам, збільшити безпеку системи та зменшити вплив можливих загроз. Основні заходи безпеки наведено нижче:

– використання надійних методів аутентифікації, таких як двофакторна аутентифікація або використання сильних паролів. Забезпечення належного контролю доступу до функцій та ресурсів системи за допомогою механізмів авторизації.

– перевірка та фільтрація введених даних з метою запобігання вразливостей, таких як SQL-ін'єкція або крос-сайтовий скриптинг. Це може включати перевірку типів даних, обмеження довжини, застосування масок або регулярних виразів.

– замість вбудовування значень в SQL-запити безпосередньо, використовуються параметризовані запити для запобігання SQL-ін'єкції. Це дозволяє відокремити дані від команд SQL та надає захист від небажаних впливів[18].

– застосування механізмів, таких як Content Security Policy (CSP), що дозволяють встановлювати правила для обмеження джерел вмісту та запобігають виконанню шкідливого коду.

– використання безпечних механізмів для керування сесіями, таких як унікальні токени сесії, обмеження строку дії сесії та захист від атак типу "сесія вкрадена".

– використання принципу найменших привілеїв та належне налаштування рольового доступу для забезпечення тільки необхідного рівня доступу до різних функцій та даних в системі.

Висновки до розділу

Розділ "Моделювання та конструювання програмного забезпечення" був присвячений детальному опису процесу розробки та конструювання проєкту "Веб-сервіс для генерації та розгортання лендингів". Під час роботи над проєктом було проведено аналіз вимог та визначено основні функціональність та характеристики системи. Далі була обрана монолітна архітектура, що складається з компонентів моделі, представлення та контролера, і реалізована за допомогою фреймворка NestJS.

У процесі розробки були використані різноманітні сторонні утиліти, бібліотеки та фреймворки, які значно спростили розробку та надали необхідні функціональні можливості. Використання фреймворку React дозволило реалізувати компонентний підхід та створити зручний інтерфейс для користувачів. Також, була використана бібліотека Axios для здійснення HTTP-запитів, що дозволило ефективно взаємодіяти з сервером.

У контексті безпеки даних було проведено аналіз потенційних загроз, ідентифікацію ризиків та розробку заходів для їх запобігання та мінімізації. Зокрема, були використані заходи, такі як валідація даних, використання параметризованих запитів до бази даних та механізми безпеки веб-програмування, такі як CSP.

В процесі роботи над проєктом було розглянуто також питання розгортання системи. Застосовано алгоритм розгортання, який включає підготовку середовища, встановлення необхідного програмного забезпечення та налаштування серверів. Для зберігання даних були використані дві бази даних - PostgreSQL для структурованих даних та Redis для тимчасових даних та кешування.

Результатом виконаною роботи є функціональний веб-сервіс для генерації та розгортання лендингів, який дозволяє користувачам зручно створювати та налаштовувати свої лендінги.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

У рамках розробки веб-сервісу для генерації та розгортання лендингів був проведений аналіз якості програмного забезпечення. Цей аналіз включав оцінку різних аспектів якості, таких як: функціональність, надійність, ефективність, зручність використання та безпека.

Один із аспектів, що був проаналізований - це функціональність веб-сервісу. Було перевірено, наскільки сервіс задовольняє функціональні вимоги, включаючи можливість створення та налаштування лендингів, збереження та відображення даних користувачів. Результати показали, що веб-сервіс успішно виконує свої функції та задовольняє вимоги користувачів.

Надійність є ще одним важливим аспектом якості програмного забезпечення. Було проведено тестування, щоб переконатися, що сервіс працездатний, стабільний та відповідає вимогам навантаження. Тестування включало перевірку обробки запитів, перевірку відповідності результатів запитів очікуваному поведінці та виявлення можливих помилок та вразливостей. Результати тестування підтвердили, що веб-сервіс є надійним та може витримувати навантаження.

Ефективність також була предметом аналізу. Оцінювалася швидкодія сервісу та його відповідь на запити користувачів. Було виконано вимірювання часу відгуку та завантаження системи для різних навантажень. Аналіз показав, що сервіс працює швидко та ефективно, а шкала його розширення дозволяє масштабувати його для вирішення зростаючих потреб користувачів.

Для забезпечення зручного використання веб-сервісу було звернуто увагу на його інтерфейс та користувацький досвід. Були проведені тестування інтерфейсу, ергономічні оцінки та збір фідбеку від користувачів.

Цей аналіз допоміг виявити можливі недоліки та вдосконалити інтерфейс для зручності користувачів[19].

Останнім аспектом аналізу була безпека веб-сервісу. Було застосовано різні заходи безпеки, такі як валідація даних, застосування параметризованих запитів до бази даних та використання механізмів безпеки веб-програмування. Аналіз показав, що веб-сервіс має високий рівень безпеки та дотримується найкращих практик безпеки даних та захисту від атак.

У ході проведеного дослідження було вирішено використовувати SonarQube для оцінки якості програмного забезпечення. На рисунку 3.1 зображено звіт SonarQube.

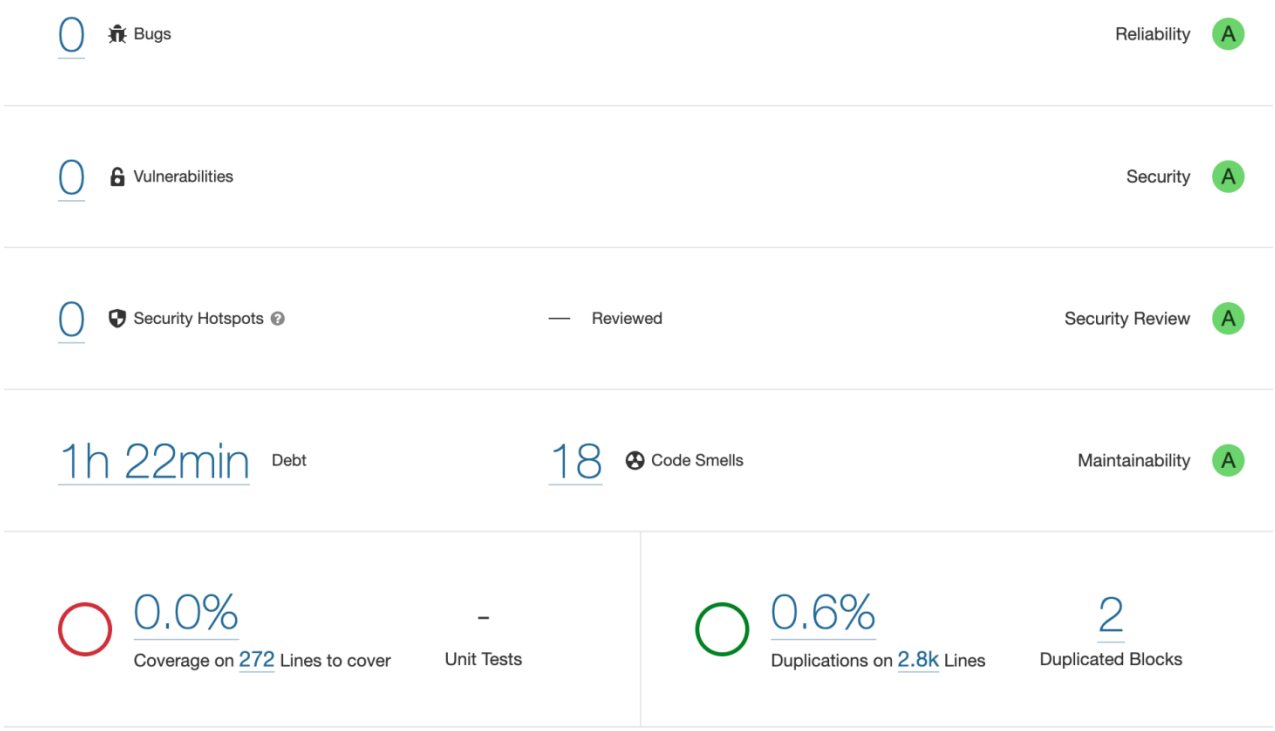


Рисунок 3.1 – Звіт SonarQube

Отже, аналіз якості програмного забезпечення підтвердив, що веб-сервіс для генерації та розгортання лендингів є функціональним, надійним, ефективним, зручним у використанні та безпечним. Результати аналізу будуть використані для подальшого вдосконалення та розвитку сервісу з метою забезпечення задоволення потреб користувачів і досягнення успіху в його використанні.

3.2 Опис процесів тестування

У даному підрозділі описано метод, який використовуються для перевірки якості розробленого веб-сервісу. Цим методом є ручне тестування, яке було проведене для програмного забезпечення "Веб-сервіс для генерації та розгортання лендингів". Детальні описи відповідних тестів наведено у таблицях 3.1 – 3.8, де вказані кроки, очікувані результати та вхідні дані для кожного тесту.

Таблиця 3.1 – Тест 1.1

Тест	Реєстрація користувача
Модуль	Реєстрація
Номер тесту	1.1
Початковий стан системи	Користувач знаходиться на сторінці реєстрації.
Вхідні данні	Ім’я, прізвище, електронна пошта, пароль
Опис проведення тесту	У відповідні поля вводяться: ім’я та прізвище користувача, валідна електронна пошта, яка до цього не була зареєстрована в системі, пароль від 8 символів, який містить хоча б з одну англійську літеру, одне число. Після цього користувач натискає кнопку «Зареєструватися».
Очікуваний результат	Користувач успішно зареєструвався та додається у систему і перенаправляється на домашню сторінку.
Фактичний результат	Користувач успішно зареєструвався та додається у систему і перенаправляється на домашню сторінку.

Таблиця 3.2 – Тест 1.2

Тест	Авторизація користувача
Модуль	Авторизація
Номер тесту	1.2

Продовження таблиці 3.2

Початковий стан системи	Користувач знаходиться на сторінці авторизації.
Вхідні данні	Електронна пошта, пароль
Опис проведення тесту	У відповідні поля вводяться: валідна електронна пошта, яка до цього вже була зареєстрована в системі та відповідний пароль. Після цього користувач натискає кнопку «Увійти».
Очікуваний результат	Користувач успішно авторизується і перенаправляється на домашню сторінку.
Фактичний результат	Користувач успішно авторизується і перенаправляється на домашню сторінку.

Таблиця 3.3 – Тест 1.3

Тест	Створення веб-сторінки
Модуль	Веб-сторінки
Номер тесту	1.3
Початковий стан системи	Користувач знаходиться на домашній сторінці свого акаунту.
Вхідні данні	-
Опис проведення тесту	Користувач натискає кнопку «+», відкривається модульне вікно зі списком доступних шаблонів. Користувач обирає потрібний шаблон та натискає на нього.
Очікуваний результат	Створюється картка шаблону на домашній сторінці користувача з короткою інформацією та датою створення.
Фактичний результат	Створюється картка шаблону на домашній сторінці користувача з короткою інформацією та датою створення.

Таблиця 3.4 – Тест 1.4

Тест	Перегляд списку створених веб-сторінок
Модуль	Веб-сторінки
Номер тесту	1.4
Початковий стан системи	Користувач знаходиться на домашній сторінці свого акаунту.
Вхідні данні	-
Опис проведення тесту	Після створення веб-сторінки, користувач бачить картку зі створеною веб-сторінкою.
Очікуваний результат	Система відображає картку створеної веб-сторінки.
Фактичний результат	Система відображає картку створеної веб-сторінки.

Таблиця 3.5 – Тест 1.5

Тест	Видалення веб-сторінки
Модуль	Веб-сторінки
Номер тесту	1.5
Початковий стан системи	Користувач знаходиться на домашній сторінці свого акаунту.
Вхідні данні	-
Опис проведення тесту	Створена як мінімум одна веб-сторінка. Якщо користувач бажає видалити веб-сторінку, він натискає кнопку «-» в правому верхньому куті картки шаблону.
Очікуваний результат	Система видаляє картку шаблону і вона більше не відображається на домашній сторінці користувача.
Фактичний результат	Система видаляє картку шаблону і вона більше не відображається на домашній сторінці користувача.

Таблиця 3.6 – Тест 1.6

Тест	Редагування веб-сторінки
Модуль	Веб-сторінки
Номер тесту	1.6
Початковий стан системи	Користувач знаходиться на домашній сторінці свого акаунту.
Вхідні данні	-
Опис проведення тесту	Створена як мінімум одна веб-сторінка. Якщо користувач бажає перейти в режим редагування веб-сторінки, він натискає зелену кнопку «Edit» в лівому нижньому куті картки шаблону.
Очікуваний результат	Система перенаправляє користувача на вибраний шаблон в режим редагування.
Фактичний результат	Система перенаправляє користувача на вибраний шаблон в режим редагування.

Таблиця 3.7 – Тест 1.7

Тест	Розгортання веб-сторінки
Модуль	Веб-сторінки
Номер тесту	1.7
Початковий стан системи	Користувач знаходиться на домашній сторінці свого акаунту.
Вхідні данні	-
Опис проведення тесту	Створена як мінімум одна веб-сторінка. Якщо користувач бажає розгорнути веб-сторінку та переглянути результат редагування своєї веб-сторінки, він натискає жовту кнопку «Expand» в правому нижньому куті картки шаблону.
Очікуваний результат	Система розгортає обраний шаблон в новому вікні та збереженими змінами.

Продовження таблиці 3.7

Фактичний результат	Система розгортає обраний шаблон в новому вікні та збереженими змінами.

Таблиця 3.8 – Тест 1.8

Тест	Вихід користувача з системи
Модуль	Вихід
Номер тесту	1.8
Початковий стан системи	Користувач знаходиться на домашній сторінці свого акаунту.
Вхідні данні	-
Опис проведення тесту	Користувач наводить курсор миші на своє ім’я, знизу з’являється кнопка «Вихід», яку користувач натискає.
Очікуваний результат	Користувач виходить зі свого акаунту зі збереженням результатів роботи користувача на своїй домашній сторінці.
Фактичний результат	Користувач виходить зі свого акаунту зі збереженням результатів роботи користувача на своїй домашній сторінці.

3.3 Опис контрольного прикладу

Контрольний приклад є важливою частиною аналізу якості програмного забезпечення, він допомагає перевірити роботу програмного забезпечення з практичної точки зору та визначити, наскільки відповідає воно поставленим вимогам.. В контексті дипломного проєкту "Веб-сервіс для генерації та розгортання лендингів" було розроблено та виконано контрольний приклад, який наведено на рисунках 3.1 – 3.10.

Неавторизований користувач знаходиться на початковій сторінці (рисунок 3.2).

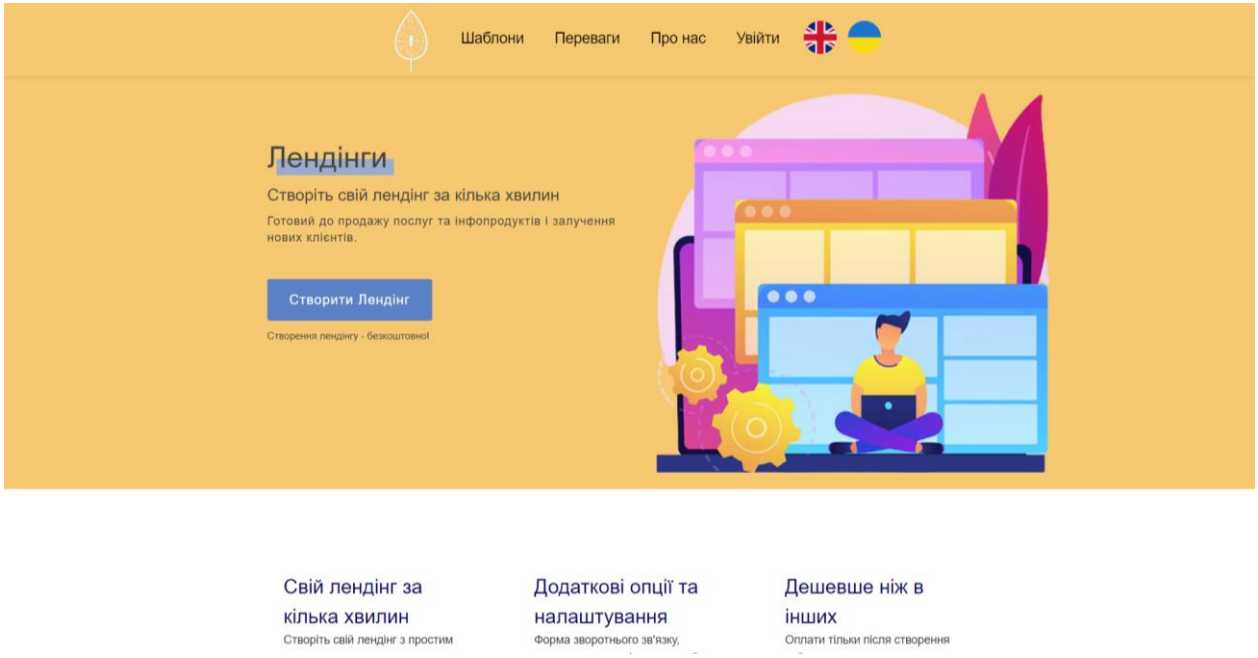


Рисунок 3.2 – Початкова сторінка

Неавторизований користувач переходить на сторінку реєстрації (рисунок 3.3).

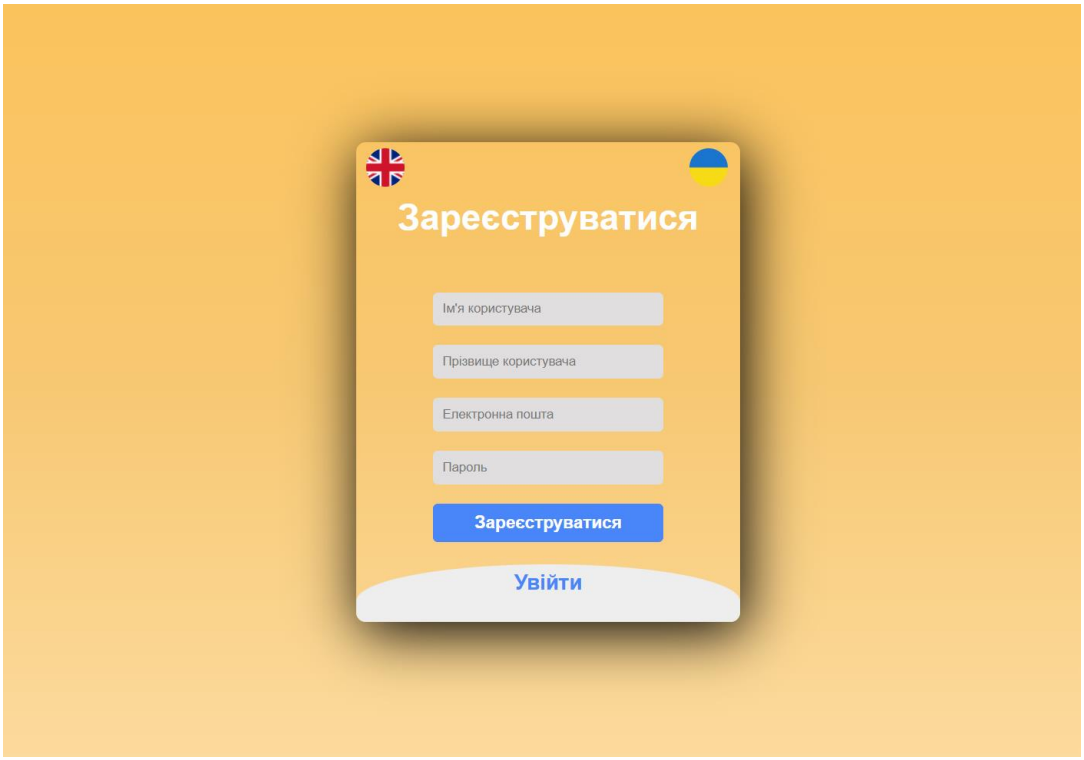


Рисунок 3.3 – Сторінка реєстрації

Неавторизований користувач може зареєструватися, якщо в нього немає облікового запису в системі, але було використано тестовий акаунт (рисунок 3.4).

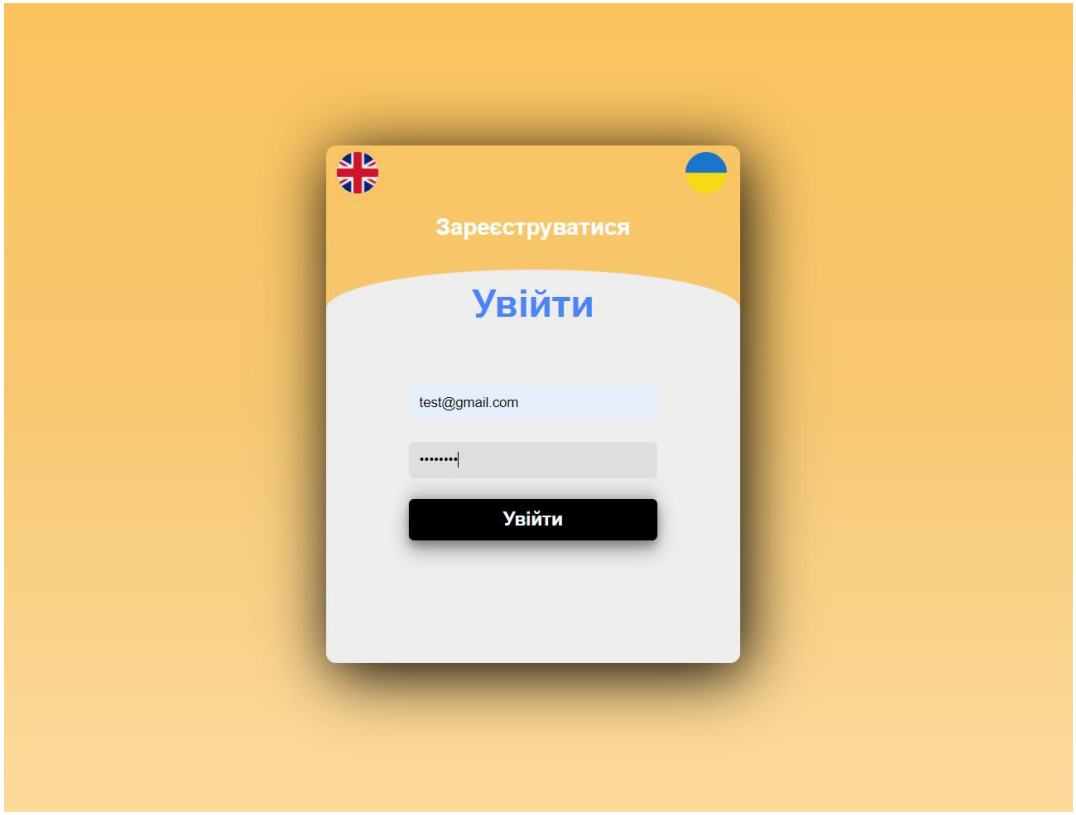


Рисунок 3.4 – Сторінка авторизації

Увійшовши до свого облікового запису, користувач знаходиться на домашній сторінці свого профілю, де може переглянути створені веб-сторінки (рисунок 3.5).

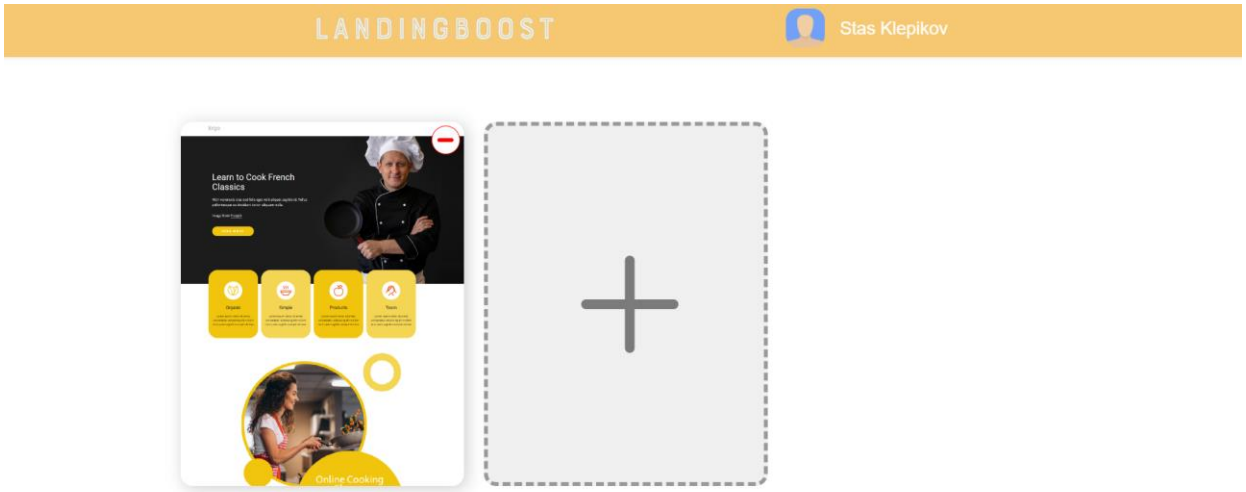


Рисунок 3.5 – Домашня сторінка профілю

Користувач може створити нову веб-сторінку натиснувши кнопку «+» та вибрати шаблон зі списку шаблонів в модальному вікні (рисунок 3.6).

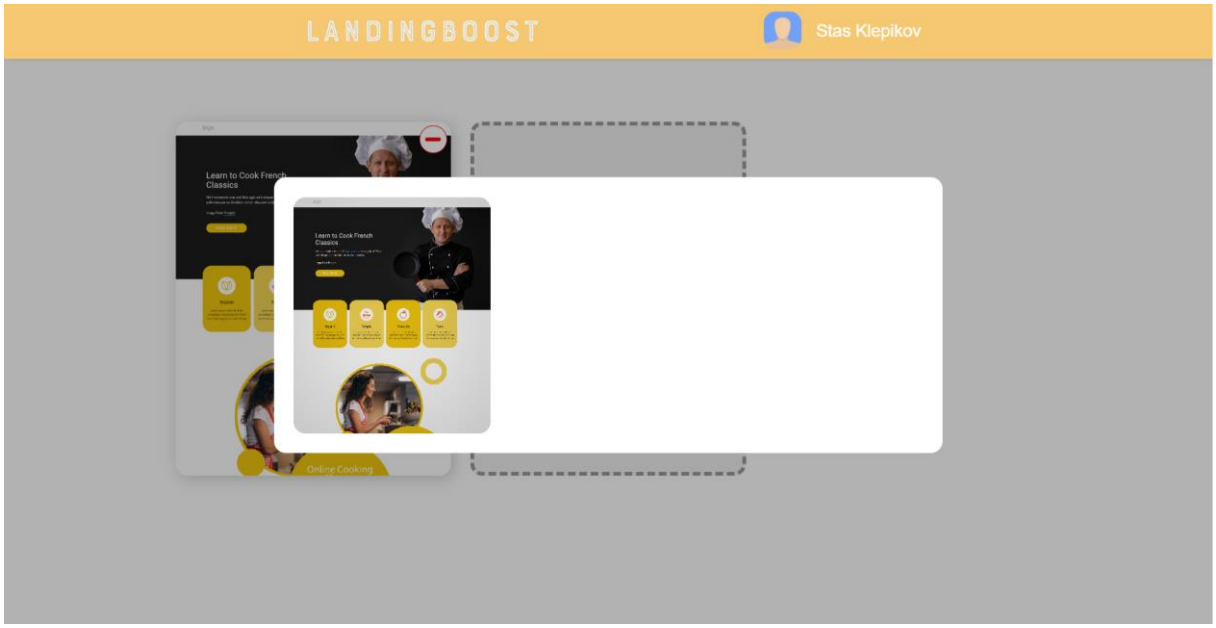


Рисунок 3.6 – Список шаблонів в модальному вікні

Після створення нової веб-сторінки, користувач може переглянути коротку інформацію про створену веб-сторінку, а також редагувати її або розгорнути (рисунок 3.7).

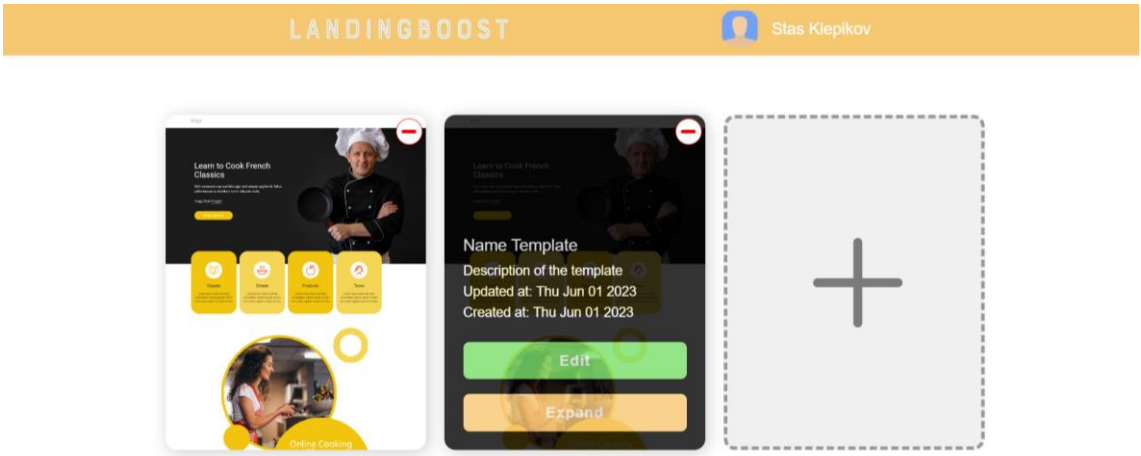


Рисунок 3.7 – Коротка інформація про шаблон, а також кнопки «Редагувати» та «Розгорнути»

Користувач може редагувати створену веб-сторінку натиснувши кнопку «Редагувати», після чого його перенаправить на створений шаблон в режимі редагування, де він зможе змінювати наповнення веб-сторінки та кольорову гаму. Відредагувавши шаблон, користувач натискає кнопку «Зберегти», щоб зберегти зміни в шаблоні (рисунок 3.8).

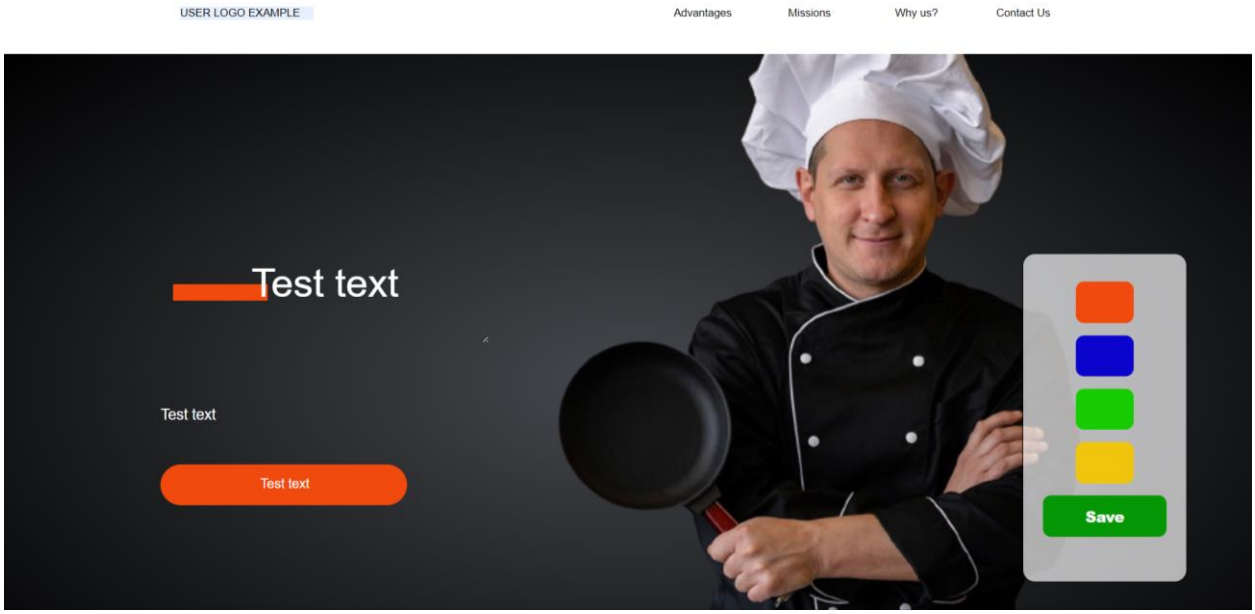


Рисунок 3.8 – Режим редагування шаблону

Після внесення змін в шаблон, користувач може розгорнути веб-сторінку та подивитися збережені зміни (рисунок 3.9).

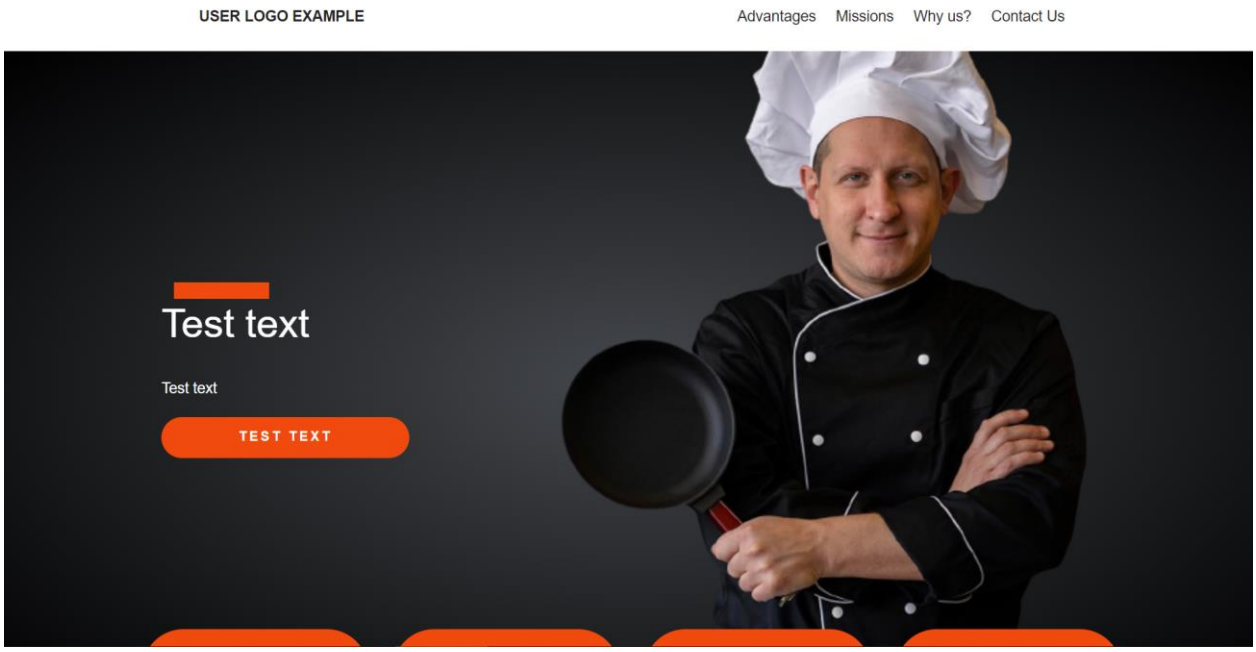


Рисунок 3.9 – Розгорнута веб-сторінка з внесеними змінами

Якщо користувачу потрібно видалити веб-сторінку, він може натиснути кнопку «-» в верхньому правому куті та веб-сторінка зникне з домашньої сторінки користувача та видалиться (рисунок 3.10).

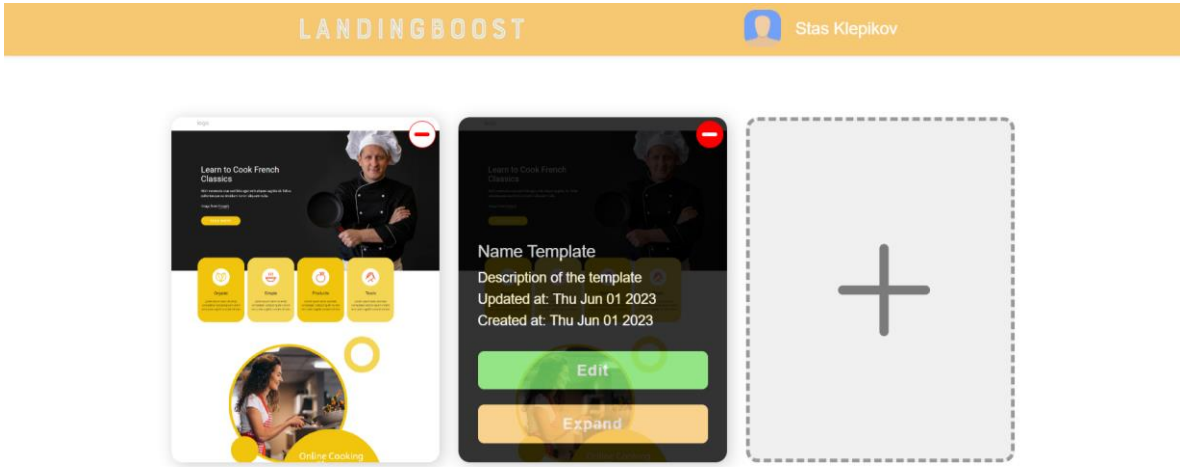


Рисунок 3.10 – Видалення веб-сторінки

Висновки до розділу

У розділі "Аналіз якості та тестування програмного забезпечення" було проведено комплексний аналіз якості веб-сервісу для генерації та розгортання лендингів. Дослідження включало аналіз якості ПЗ, опис процесів тестування та створення контрольного прикладу. На основі проведеного аналізу можна зробити наступні висновки:

Аналіз якості ПЗ показав, що веб-сервіс має задовільний рівень якості, але потребує деяких поліпшень. Виявлені метрики, такі як функціональність, надійність, ефективність, безпека, допомогли виявити сильні та слабкі сторони програмного забезпечення. Були визначені конкретні заходи для покращення якості ПЗ, включаючи оптимізацію коду, стандартизацію, та забезпечення достатнього рівня безпеки.

Опис процесів тестування включав виконання ручних тестів згідно з визначеними тестовими сценаріями. Цей процес дозволив виявити недоліки у функціоналі веб-сервісу. Проведене тестування також допомогло підтвердити відповідність програмного забезпечення вимогам та забезпечити високу якість виконання функцій та задач.

Опис контрольного прикладу дав можливість провести більш глибокий аналіз конкретної функціональності платформи. Цей приклад дозволив виявити та виправити деякі дефекти у роботі програмного забезпечення. Він також вказав на необхідність подальшого розвитку та вдосконалення деяких аспектів функціоналу та користувацького досвіду.

В цілому, проведений аналіз якості та тестування програмного забезпечення дозволив виявити й усунути помилки та проблеми, підвищити рівень якості веб-сервісу та забезпечити високу задоволеність користувачів. Рекомендації та пропозиції щодо подальшого вдосконалення програмного забезпечення були враховані та втілені у процесі його розвитку. Завдяки цим заходам забезпечення якості, веб-сервіс став більш стабільним, надійним та функціональним, що відповідає потребам та очікуванням користувачів.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для розгортання програмного забезпечення було використано платформи Heroku та AWS.

Переваги Heroku:

– Heroku надає зручний та інтуїтивно зрозумілий інтерфейс, що дозволяє легко налаштовувати та розгортати програмне забезпечення.

– Heroku дозволяє легко масштабувати програмне забезпечення відповідно до зростання обсягу користувачів та навантаження. Платформа забезпечує автоматичне масштабування та балансування навантаження, що дозволяє зберігати веб-сервіс доступним та швидким.

– Heroku має широкий набір інтегрованих сервісів, таких як бази даних, системи кешування, логування та інші. Це дозволяє легко підключати й налаштовувати різні компоненти системи та забезпечувати їх взаємодію[20].

Переваги AWS:

– AWS має високі стандарти безпеки та надійності, забезпечуючи захист даних та надійну роботу інфраструктури. Платформа пропонує різноманітні сервіси для забезпечення безпеки, такі як мережеві файрволи, шифрування даних, контроль доступу та інше.

– AWS надає багато різноманітних сервісів, які можна використовувати в проєкті. Наприклад, Amazon RDS для керування базами даних та Amazon S3 для зберігання медіа-файлів.

– AWS надає гнучкі можливості масштабування інфраструктури залежно від зростання обсягу користувачів та навантаження. Вона забезпечує автоматичне масштабування ресурсів, що дозволяє забезпечувати високу доступність та продуктивність веб-сервісу[21].

Процес розгортання включав декілька важливих кроків, таких як:

Створення бази даних PostgreSQL на Amazon RDS, після чого налаштування необхідних параметрів, таких як доступ, автентифікація та резервне копіювання, для забезпечення надійності та безпеки даних (рисунок 4.1).

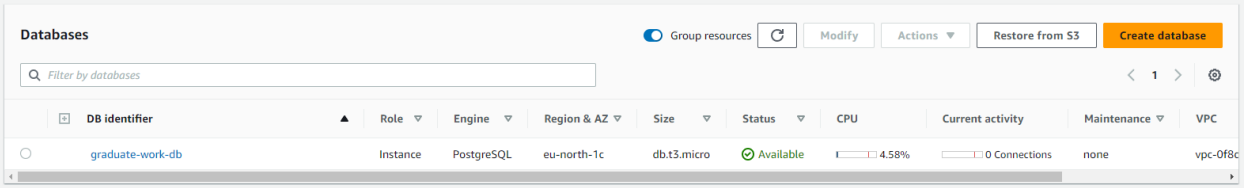


Рисунок 4.1 – Створена база даних PostgreSQL на Amazon RDS

Створення та налаштування сервісу Redis Cloud для зберігання тимчасових даних, кешування результатів запитів та швидкого доступу до ресурсів (рисунок 4.2). Налаштування включало параметри безпеки, доступу та розміщення ресурсу.

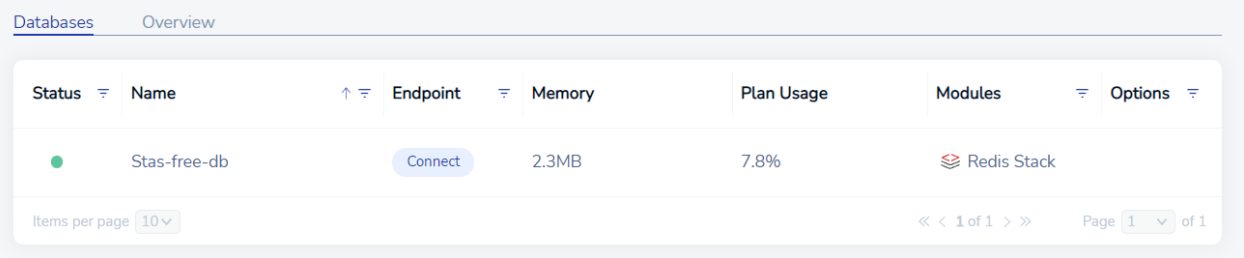


Рисунок 4.2 – Створена база даних Redis

Для зберігання та керування медіа-файлами було налаштовано та використано сервіс AWS S3. Налаштування включало створення бакету, налаштування доступу та інтеграцію з програмним забезпеченням (рисунок 4.3).

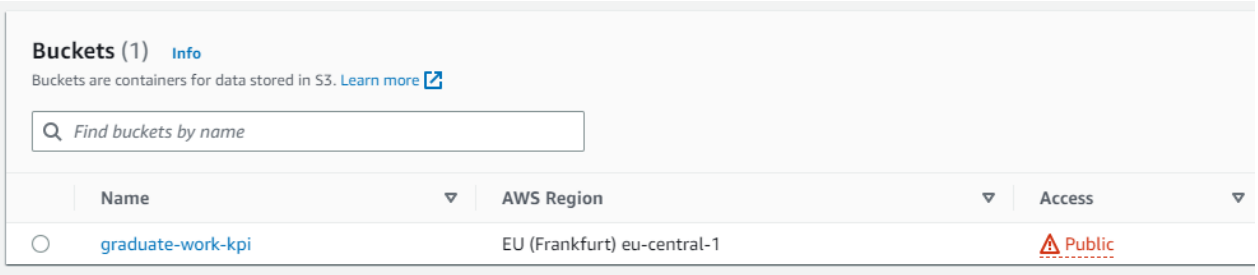


Рисунок 4.3 – Створений бакет на платформі AWS S3

Серверна та клієнтська частини були розгорнуті на платформі Heroku, яка надає зручні механізми для розгортання, масштабування та керування програмним забезпеченням. Код програми був завантажений на Heroku, де

було виконано налаштування залежностей та автоматично виконувалась процедура розгортання (рисунок 4.4).

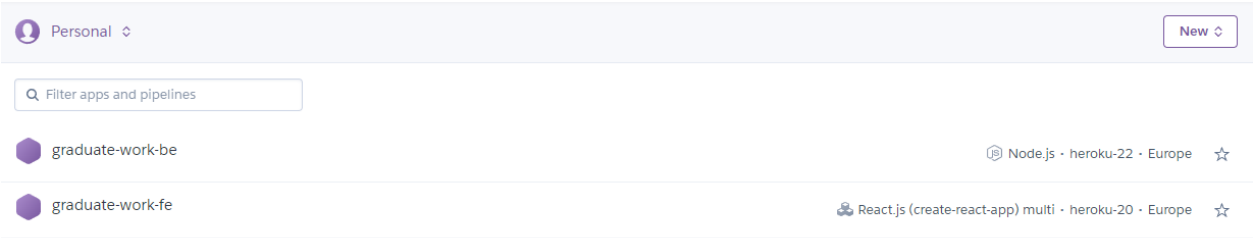


Рисунок 4.4 – Серверна та клієнтська частини на платформі Heroku

У результаті цих кроків програмне забезпечення було успішно розгорнуто на платформах Heroku та Amazon RDS, забезпечуючи стабільну роботу, доступ до бази даних та інших необхідних ресурсів.

4.2 Підтримка програмного забезпечення

Підтримка програмного забезпечення є важливим етапом у життєвому циклі будь-якого проекту. Вона включає в себе ряд дій та процесів, спрямованих на забезпечення правильної роботи, надійності та ефективності програмного продукту після його впровадження. Один з ключових елементів підтримки програмного забезпечення полягає у наданні клієнтам оновлень системи. Підтримка програмного забезпечення включає розробку нових версій системи, тестування, реліз, розгортання та надання клієнтських оновлень. Працюючи над розширенням функціональності, виправленням помилок та вдосконаленням продукту, зміни вносяться до кодової бази програмного забезпечення. Перед випуском нової версії вона проходить процес тестування для перевірки наявності помилок та забезпечення стабільної роботи, включаючи різні види тестів. Після успішного тестування нова версія готова до релізу, який включає пакування програмного забезпечення та розгортання на сервер звідки вже потрапить користувачам. Розгортання нової версії може відбуватись шляхом оновлення встановленого ПЗ на серверах або постачання її клієнтам для встановлення на їх комп'ютери або пристрої. Клієнти отримують оновлення через автоматичне оновлення.

Висновки до розділу

У розділі "Впровадження та супровід програмного забезпечення" було проведено основні кроки для ефективного впровадження та підтримки програмного забезпечення. Було проаналізовано та по результатам вирішено використовувати платформу Heroku для розгортання клієнтської та серверної частин програмного забезпечення. Ця платформа має відповідні можливості та інструменти, які підходять для потреб проєкту "Веб-сервіс для генерації та розгортання лендингів". Це дозволяє забезпечити доступ до веб-сервісу для користувачів через Інтернет. База даних PostgreSQL була створена та налаштована на платформі Amazon RDS. Це забезпечує надійне зберігання та доступ до даних, необхідних для функціонування веб-сервісу. Також було створено та налаштовано Redis Cloud, що забезпечує швидке та ефективне кешування даних, покращуючи продуктивність системи. Для зберігання медіа-файлів, що використовуються в лендингах було створено та налаштовано бакет на Amazon S3.

В результаті проведених робіт з впровадження та супроводу програмного забезпечення було створено працездатний веб-сервіс для генерації та розгортання лендингів. Використання платформи Heroku та інших сервісів AWS дозволяє забезпечити стабільну роботу системи, швидку доставку контенту та ефективне управління даними. Підтримка програмного забезпечення включає розробку нових версій, тестування, реліз, розгортання та надання оновлень клієнтам, забезпечуючи сталу еволюцію та підтримку функціональності системи.

ВИСНОВКИ

Дипломна робота присвячена розробці веб-сервісу для генерації та розгортання лендингів, за допомогою якого користувачі можуть з легкістю створювати свої веб-сторінки на основі існуючих шаблонів.

В першому розділі проведено аналіз вимог до програмного забезпечення. Визначені ключові цілі та функціональні вимоги, включаючи можливості платформи, інтерфейс користувача, безпеку даних та приватність. Результати аналізу показали, що проєкт вимагатиме розробки системи з урахуванням встановлених вимог та забезпеченням зручного інтерфейсу та надійності. Метою проєкту є прискорення процесу створення та розгортання лендингів з метою вирішення проблеми складності і затрат, пов'язаних з цими процесами. Веб-сервіс прагне надати користувачам простий та ефективний інструмент.

Другий розділ детально описує процес розробки та конструювання веб-сервісу. Аналіз вимог та вибір монолітної архітектури були початковими кроками. Застосування фреймворку NestJS та бібліотеки React дозволило створити зручний інтерфейс та ефективно взаємодіяти з сервером. Проведено аналіз безпеки даних та впроваджено заходи для запобігання загрозам та мінімізації ризиків. Також вирішено питання зберігання даних, використовуючи PostgreSQL та Redis. Результатом роботи є функціональний веб-сервіс, що дозволяє зручно створювати та налаштовувати лендінги.

В третьому розділі було проведено комплексний аналіз якості веб-сервісу для генерації та розгортання лендингів, виявивши сильні та слабкі сторони ПЗ. Заходи для покращення якості були визначені на основі аналізу метрик, а процес тестування виявив недоліки та підтвердив відповідність вимогам. Контрольний приклад допоміг покращити функціональність та виявити напрямки розвитку. В результаті, програмне забезпечення стало більш стабільним та задовольнило потреби користувачів, завдяки заходам

забезпечення якості та врахуванню рекомендацій для подальшого вдосконалення.

В четвертому розділі були виконані основні кроки для ефективного впровадження та підтримки веб-сервісу. Використання платформи Heroku для розгортання, Amazon RDS для бази даних PostgreSQL та Redis Cloud для кешування даних забезпечують стабільну роботу системи та надійне зберігання і доступ до даних. Крім того, Amazon S3 використовується для зберігання медіа-файлів. В результаті цих дій було створено працездатний веб-сервіс, який забезпечує швидку доставку контенту та ефективне управління даними. Підтримка програмного забезпечення включає розробку нових версій, тестування, реліз, розгортання та надання оновлень користувачам, забезпечуючи сталу еволюцію та підтримку функціональності системи.

При вирішенні поставлених задач були отримані наступні висновки:

– розробка веб-сервісу для генерації та розгортання лендингів є доцільною та перспективною. У складний період війни, бізнес робить значний внесок в економіку країни. Через відтік людей, малому бізнесу складніше виживати, тому було прийнято рішення розробити не дорогий, простий інструмент. Використання такого сервісу дозволяє автоматизувати процес створення лендингів і зменшує зусилля, час та ресурси, необхідні для їх розгортання.

– розроблений веб-сервіс ефективно впроваджує функціональність генерації лендингів з використанням шаблонів. Користувачам надається можливість створити та налаштувати лендинг, використовуючи доступні інструменти та функціонал сервісу.

– проведене тестування веб-сервісу показало його стабільну та надійну роботу. Були виявлені та виправлені потенційні помилки та недоліки, що дозволяє забезпечити якість та надійність сервісу для його користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Aicontentfy. *The Power of Landing Pages in Maximizing Content Distribution.* URL: https://aicontentfy.com/en/blog/power-of-landing-pages-in-maximizing-content-distribution.
2. Hotjar. *Understanding the basics of landing page optimization.* URL: https://www.hotjar.com/landing-page-optimization/.
3. Amazon Web Servises. *What is Amazon EC2?* URL: https://docs.aws.amazon.com/AWSEC2 /latest/UserGuide/concepts.html.
4. Crazyegg. *The Ultimate Guide to Landing Page Optimization*. URL: https://www.crazyegg.com/blog/landing-page-optimization/.
5. Tiny.cloud. *What is a WYSIWYG editor?* URL: https://www. tiny.cloud/blog/ wysiwyg-html-editor/.
6. Planetcrust. *What is a Drag-and-Drop Editor?* URL: https://www.planetcrust.com/what-is-a-drag-and-drop-editor?utm_campaign=blog.
7. React. *Official documentation.* URL: https://legacy.reactjs.org/docs/ getting-started.html.
8. NestJS. *Official documentation.* URL: https://docs.nestjs.com/.
9. I18next. *Official documentation.* URL: https://react.i18next.com/.
10. React Router. *Official documentation.* URL: https://reactrouter. com/en/main.
11. PostgreSQL. *What is PostgreSQL?* URL: https://www.postgresql. org/about/.
12. Redis. *Official documentation.* URL: https://redis.io/docs/.
13. WordPress. URL: https://wordpress.com/ru/features/.
14. Wix. URL: https://uk.wix.com/.
15. Unbounce. URL: https://unbounce.com/.
16. Arctype. *How to build a NestJS MVC application.* URL: https://arctype.com/blog/nestjs-mvc/.

17. Freecodecamp. *Axios With React: The Definitive Guide*. URL: https://www.freecodecamp.org/news/how-to-use-axios-with-react/.

18. Cheatsheetseries. *SQL Injection Prevention Cheat Sheet.* URL: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html.

19. Markettailor. *The role of usability in landing page*. URL: https://www.markettailor.io/blog/role-of-usability-in-landing-page-optimization.

20. Heroku. *Official documentation.* URL: https://devcenter.heroku.com/.

21. Amazon Web Servises. *Official documentation.* URL: https://docs.aws. amazon.com/rds/?icmpid=docs_homepage_featuredsvcs.

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015405711

Дата перевірки:
03.06.2023 13:33:41 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
03.06.2023 13:39:59 EEST

ID користувача:
76913

Назва документа: **ІТ-93_Клепіков_ПЗ**

Кількість сторінок: **60** **Кількість слів:** **10887** **Кількість символів:** **87727** **Розмір файлу:** **10.44 MB** **ID файлу:** **1015069359**

8.34%
Схожість

Найбільша схожість: **3.28% з джерелом з Бібліотеки** (**ID файлу:** **1015042084**)

3.01% Джерела з Інтернету **165** ..... **Сторінка** 62

8.13% Джерела з Бібліотеки **400** ..... **Сторінка** 65

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

ДОДАТОК Б DOCKERFILE ДЛЯ РОЗГОРТАННЯ СЕРВЕРНОЇ ЧАСТИНИ

```
FROM node:lts-buster-slim

WORKDIR /app

COPY package.json package-lock.json ./

RUN npm ci

COPY . .

RUN npm run build

ADD ./docker-entrypoint.sh /usr/local/bin/

RUN chmod +x /usr/local/bin/docker-entrypoint.sh

ENV PORT 3000

EXPOSE $PORT

ENTRYPOINT ["docker-entrypoint.sh"]
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

ВЕБ-СЕРВІС ДЛЯ ГЕНЕРАЦІЇ ТА РОЗГОРТАННЯ ЛЕНДИНГІВ

Текст програми

КПІ.ІТ-9314.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Микола ХРАМЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Станіслав КЛЕПІКОВ

Київ – 2023

Файл App.jsx

```
import './App.scss';
import './components/fonts.scss'
import { Routes, Route } from 'react-router-dom';
import React, { useState } from "react";
import Registration from './components/registration/Registration';
import Page from './components/main_page/Page';
import Loading from './components/loading/Loading';
import Home from './components/user_page/home/Home';
import TemlateCook from
'./components/user_page/templates/template_cook/Temlate_cook';

function App() {
    const [isLoading, setIsLoading] = useState(true);
    const timeout = setTimeout(() => setIsLoading(false), 1000);
    return (
        <div className="wrapper">
            {isLoading ? <Loading /> :
                <Routes>
                    <Route path='/' element={<Page />} />
                    <Route path='registration' element={<Registration />} />
                    <Route path="home" element={<Home />} />
                    <Route path="template_cook/:id" element={<TemlateCook />}
/>
                </Routes>}
        </div>
  );
}

export default App;
```

Файл Page.jsx

```
import './Page.scss';
import Header from '../header/Header';
import Footer from '../footer/Footer';
import Description from '../content/description/Description';
import Advantages from '../content/advantages/Advantages';
import About from '../content/about/About';
import Banner from '../content/banner/Banner'
```

```
function Page() {
    console.log(process.env.REACT_APP_API_URL)
    return (
        <div className="page">
            <Header />
            <Description />
            <Advantages />
            <About />
            <Banner/>
            <Footer />
        </div>
  );
}

export default Page;
```

Файл Registration.jsx

```
import axios from 'axios';
import './Registration.scss';
import { useTranslation } from 'react-i18next';
import { useNavigate } from "react-router-dom";
import React from 'react';

function Registration() {

    const { t, i18n } = useTranslation();
    const changeLanguage = (language) => {
        i18n.changeLanguage(language);
    };

    const [name, setName] = React.useState();
    const [surname, setSurname] = React.useState();
    const [email, setEmail] = React.useState();
    const [password, setPassword] = React.useState();

    const [nameDirty, setNameDirty] = React.useState(false);
    const [surnameDirty, setSurnameDirty] = React.useState(false);
    const [emailDirty, setEmailDirty] = React.useState(false);
    const [passwordDirty, setPasswordDirty] = React.useState(false);
```

```
    const [nameError, setNameError] =
React.useState(t("registration.input_name_error_empty"));
    const [surnameError, setSurnameError] =
React.useState(t("registration.input_surname_error_empty"));
    const [emailError, setEmailError] =
React.useState(t("registration.input_email_error_empty"));
    const [passwordError, setPasswordError] =
React.useState(t("registration.input_password_error_empty"));

    const [inputName, setInputName] = React.useState('');
    const [inputSurname, setInputSurname] = React.useState('');
    const [inputEmailReg, setInputEmailReg] = React.useState('');
    const [inputPasswordReg, setInputPasswordReg] = React.useState('');
    const [inputEmailLog, setInputEmailLog] = React.useState('');
    const [inputPasswordLog, setInputPasswordLog] = React.useState('');
    const navigate = useNavigate();

    function login(email, password) {
        return axios.post(`${process.env.REACT_APP_API_URL}/api/v1/sign-in`,
{
            email,
            password
        })
        .then(({ data }) => {
            localStorage.setItem('auth', JSON.stringify(data));
            navigate("/home");
        }).catch(err => {
            console.log('ERROR!');
        });
    }

    function handleLogin(event) {
        event.preventDefault();
        login(inputEmailLog, inputPasswordLog);
    }

    function handleRegister(event) {
        event.preventDefault();
        axios.post(`${process.env.REACT_APP_API_URL}/api/v1/sign-up`, {
            name: inputName,
            surname: inputSurname,
            email: inputEmailReg,
            password: inputPasswordReg
```

```
        })
        .then(({ data }) => {
            login(inputEmailReg, inputPasswordReg);
        }).catch(err => {
            console.log('ERROR!');
        });
    }

    const blurHandle = (event) => {
        switch (event.target.name) {
            case 'email':
                setEmailDirty(true)
                break;
            case 'pswd':
                setPasswordDirty(true);
                break;
            case 'name':
                setNameDirty(true);
                break;
            case 'surname':
                setSurnameDirty(true);
                break;
            default:
        }
    }
    const nameHandler = (event) => {
        setName(event.target.value);
        if (event.target.value.length < 3 || event.target.value.length > 64)
{
            setNameError(t("registration.input_name_error"));
            if (!event.target.value) {
                setNameError(t("registration.input_name_error_empty"));
            };
        }
        else {
            setNameError();
        };
    }
    const surnameHandler = (event) => {
        setSurname(event.target.value);
        if (event.target.value.length < 3 || event.target.value.length > 64)
{
            setSurnameError(t("registration.input_surname_error"));
```

```
            if (!event.target.value) {
                setSurnameError(t("registration.input_surname_error_empty"));
            };
        }
        else {
            setSurnameError();
        };
     }

    const emailHandler = (event) => {
        setEmail(event.target.value);
        const re = /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-
zA-Z0-9-]+)*$/;
        if (!re.test(String(event.target.value).toLowerCase())) {
            setEmailError(t("registration.input_email_error"))
        }
        else {
            setEmailError();
        }
    }

    const passwordHandler = (event) => {
        setPassword(event.target.value);
        if (event.target.value.length < 4 || event.target.value.length > 64)
{
            setPasswordError(t("registration.input_password_error"));
            if (!event.target.value) {
                setPasswordError(t("registration.input_password_error_empty")
);
            };
        }
        else {
            setPasswordError();
        };
    }

    return (
        <div className='registration'>
            <div className='registration__container'>
                <input type="checkbox" id="chk" aria-hidden="true"></input>
                <button className='btn en' onClick={() =>
changeLanguage("en")}></button>
```

```
                <button className='btn ua' onClick={() =>
changeLanguage("ua")}></button>
                <div className="registration__signup">
                    <form onSubmit={handleRegister}>
                        <label className='registration__lbl' for="chk" aria-
hidden="true">{t("registration.sign_up")}</label>
                        <input className='registration__input' type="text"
value={name} name="name" placeholder={t("registration.input_name")}
required="" onChange={(event) => { nameHandler(event);
setInputName(event.target.value) }} onBlur={event =>
blurHandle(event)}></input>
                        {(nameDirty && nameError) && <div
className='registration__error'>{ nameError}</div>}

                        <input className='registration__input' type="text"
value={surname} name="surname" placeholder={t("registration.input_surname")}
required="" onChange={(event) => { surnameHandler(event);
setInputSurname(event.target.value) }} onBlur={event =>
blurHandle(event)}></input>
                        {(surnameDirty && surnameError) && <div
className='registration__error'>{ surnameError}</div>}

                        <input className='registration__input' value={email}
type="email" name="email" placeholder={t("registration.input_email")}
required="" onChange={(event) => { emailHandler(event);
setInputEmailReg(event.target.value) }} onBlur={event =>
blurHandle(event)}></input>
                        {(emailDirty && emailError) && <div
className='registration__error'>{ emailError}</div>}

                        <input className='registration__input'
value={password} id='pswd__input' type="password" name="pswd"
placeholder={t("registration.input_password")} required="" onChange={(event)
=> { passwordHandler(event); setInputPasswordReg(event.target.value) }}
onBlur={event => blurHandle(event)}></input>
                        {(passwordDirty && passwordError) && <div
className='registration__error'>{ passwordError}</div>}
                        <button
className='registration__btn'>{t("registration.sign_up")}</button>
                    </form>
                </div>
                <div className="registration__login">
                    <form onSubmit={handleLogin}>
```

```
                        <label className='registration__lbl' for="chk" aria-
hidden="true">{t("registration.login")}</label>
                        <input className='registration__input' type="email"
value={email} name="email" placeholder={t("registration.input_email")}
required="" onChange={(event) => { emailHandler(event);
setInputEmailLog(event.target.value) }} onBlur={event =>
blurHandle(event)}></input>
                        {(emailDirty && emailError) && <div
className='registration__error'>{ emailError}</div>}

                        <input className='registration__input'
id='pswd__input' type="password" value={password} name="pswd"
placeholder={t("registration.input_pswd")} required="" onChange={(event) => {
passwordHandler(event); setInputPasswordLog(event.target.value) }}
onBlur={event => blurHandle(event)}></input>
                        {(passwordDirty && passwordError) && <div
className='registration__error'>{ passwordError}</div>}
                        <button
className='registration__btn'>{t("registration.login")}</button>
                    </form>
                </div>
            </div>
        </div>
    );
}

export default Registration;
```

Файл Add_card.jsx

```
import './Add_card.scss';
import React from 'react';
import Modal from '../modal/Modal';

function Add_card() {

    const [modalActive, setModalActive] = React.useState();

    return (
        <>
            <button id='add-card__btn' onClick={() => setModalActive(true)}>
                <div className='add-card'>
```

```
                <div className="add-card__plus">
                    <span></span>
                    <span id='add-card__span'></span>
                </div>
                </div>
            </button>
            <Modal active = {modalActive} setActive = {setModalActive} />
        </>

    );
}

export default Add_card;
```

Файл Card.jsx

```
import './Card.scss';
import axios from 'axios';
import previewtemplate_1 from '../../assets/images/previewtemplate_1.png';
import { Link } from 'react-router-dom';
import { useEffect, useState } from 'react';

function Card({ template }) {

    async function showTemplate() {
        const auth = JSON.parse(localStorage.getItem('auth'));
        return
axios.get(`${process.env.REACT_APP_API_URL}/api/v1/templates/link/${template.
id}`,
            {
                headers: {
                    Authorization: `Bearer ${auth.tokens.accessToken}`
                }
            })
            .then(({ data }) => {
                window.open(data.data, '_blank', 'noreferrer')
            }).catch(err => {
                console.log('ERROR!');
            });
    }

    async function deleteTemplate() {
```

```
        const auth = JSON.parse(localStorage.getItem('auth'));
        return
axios.delete(`${process.env.REACT_APP_API_URL}/api/v1/templates/${template.id
}`,
            {
                headers: {
                    Authorization: `Bearer ${auth.tokens.accessToken}`
                }
            })
            .then(({ data }) => {
            }).catch(err => {
                console.log('ERROR!');
            });
    }

    return (
        <>
            <div className="card">
                <button className='card__delete' onClick={() => {
deleteTemplate() }}><span></span></button>
                <img src={previewtemplate_1} alt="template-preview" />
                    <div className="card__text">
                        <h2>Name Template</h2>
                    <p>Description of the template </p>
                    <p>Updated at: { new Date
(template.updatedAt).toDateString()}</p>
                    <p>Created at: { new Date
(template.createdAt).toDateString()}</p>
                    <Link to={`/template_cook/${template.id}`}>
                        <button className='edit__btn'>Edit</button>
                    </Link>
                    <button className='show__btn' onClick={() => {
showTemplate() } }>Expand</button>
                    </div>

            </div>
        </>
    );
}

export default Card;
```

Файл Home.jsx

```
import './Home.scss';
import User_header from '../user_header/User_header'
import User_body from '../user_body/User_body'
import React from 'react';

function Home() {

    return (
        <div className='home'>
            <User_header />
            <User_body/>
        </div>
    );
}

export default Home;
```

Файл Modal.jsx

```
import './Modal.scss';
import axios from 'axios';
import React, { useState } from 'react';

function Modal({ active, setActive }) {

    async function createTemplate(name) {
        const auth = JSON.parse(localStorage.getItem('auth'));
        return
axios.post(`${process.env.REACT_APP_API_URL}/api/v1/templates`, {
            name
        },
            {
                headers: {
                    Authorization: `Bearer ${auth.tokens.accessToken}`
                }
            }
        )
            .then(({ data }) => {
                setActive(false);
```

```
            }).catch(err => {
                console.log('ERROR!');
            });
    }

    return (
        <div className={active ? "modal active": "modal"} onClick={() =>
setActive(false)}>
            <div className={active ? "modal__content active":
"modal__content"} onClick={e => e.stopPropagation()}>
                <div className='modal__btn'onClick={() =>
createTemplate('cook')}>
                    <div className="modal__slide">
                        <h5>CourseTemplate</h5>
                        <p>Description CourseTemplate</p>
                    </div>
                </div>
            </div>
        </div>
    );
}

export default Modal;
```

Файл User_body.jsx

```
import './User_body.scss';
import React, { useEffect} from 'react';
import axios from 'axios';
import Card from '../card/Card';
import Add_card from '../add_temlate_card/Add_card';

function User_body() {

    const [templates, setTemplates] = React.useState([]);

    async function getTemplates() {
        const auth = JSON.parse(localStorage.getItem('auth'));
        return axios.get(`${process.env.REACT_APP_API_URL}/api/v1/templates`,
{
            headers: {
                Authorization: `Bearer ${auth.tokens.accessToken}`
```

```
            }
        })
        .then(({ data }) => {
            setTemplates(data.data);
        }).catch(err => {
            console.log('ERROR!');
        });
    }

    useEffect(() => { getTemplates()} );

    return (
        <div className='user-body'>
            <div className="user-body__container">
                {templates.map((template) => (<Card template={template} />))}
                <Add_card />
            </div>
        </div>
    );
}

export default User_body;
```

Файл User_header.jsx

```
import './User_header.scss';
import logo_text from '../../assets/images/logo_text.png';
import user_icon from '../../assets/images/icons/user_icon.png';
import { useTranslation } from 'react-i18next';
import { useNavigate } from "react-router-dom";

function User_header() {

    const { t, i18n } = useTranslation();

    const navigate = useNavigate();
    const authJSON = localStorage.getItem('auth');
    const auth = JSON.parse(authJSON);

    function logOut() {

        localStorage.removeItem('auth');
```

```
        navigate("/");
    }

        return (
            <div className='user-header'>
                <div className="user-header__logo-container">
                    <img className="user-header__logo" src={logo_text}
alt="logo" />
                </div>
                <nav className="user-header__menu">
                    <img src={user_icon} alt='user_icon'/>
                    <button className='user-header__user'>{auth.user.name}
{auth.user.surname}</button>
                    <div className='user-header__sub-menu'>
                        <button className='user-header__btn'
onClick={logOut}>Exit</button>
                    </div>
                </nav>
            </div>
        );
    }

export default User_header;
```

Файл 18n.js

```
import i18n from 'i18next';
import Backend from 'i18next-http-backend';
import LanguageDetector from 'i18next-browser-languagedetector';
import { initReactI18next } from 'react-i18next';

i18n.use(Backend).use(LanguageDetector).use(initReactI18next).init({
  fallbackLng: 'ua',
  debug: true,
  detection: {
    order: ['queryString', 'cookie'],
    cache: ['cookie']
  },
  interpolation: {
    escapeValue: false
  }
})
```

```
export default i18n;
```

Файл s3-manager.service.ts

```
import { DeleteObjectOutput } from 'aws-sdk/clients/s3';
import { PromiseResult } from 'aws-sdk/lib/request';
import { AWSError, S3 } from 'aws-sdk';
import { ConfigType } from '@nestjs/config';
import { Inject, Injectable } from '@nestjs/common';
import InjectAwsService from '@v1/aws/decorators/inject-aws-
service.decorator';
import {
  awsNamespaceKey,
  awsNamespace,
} from 'src/modules/app/config/aws.config';

@Injectable()
export default class S3ManagerService {
  constructor(
    @InjectAwsService(S3) private readonly s3: S3,
    @Inject(awsNamespaceKey) private readonly options:
ConfigType<awsNamespace>,
  ) {}

  public async uploadFile(
    key: string,
    buffer: Buffer,
    mimetype: string,
  ): Promise<{ link: string }> {
    await this.s3
      .putObject({
        Bucket: this.options.bucket,
        Key: key,
        Body: buffer,
        ContentType: mimetype,
        CacheControl: 'max-age=86400',
      })
      .promise();

    return {
      link: this.getSignedUrlByKey(key, false),
```

```
    };
  }

  public getSignedUrlByKey(
    key: string,
    forceDownload = false,
    expires: number = 2 * 60 * 60,
  ): string {
    return this.s3.getSignedUrl('getObject', {
      Bucket: this.options.bucket,
      Key: key,
      Expires: expires,
      ResponseContentDisposition: forceDownload
        ? `attachment;filename=${encodeURIComponent(key)}`
        : undefined,
    });
  }

  public async deleteFile(
    key: string,
  ): Promise<PromiseResult<DeleteObjectOutput, AWSError>> {
    return this.s3
      .deleteObject({
        Bucket: this.options.bucket,
        Key: key,
      })
      .promise();
  }
}
```

Файл redis.service.ts

```
import { createClient } from 'redis';
import { RedisClientOptions } from '@redis/client';
import { Injectable, OnModuleDestroy, OnModuleInit } from '@nestjs/common';

@Injectable()
export default class RedisService implements OnModuleInit, OnModuleDestroy {
  private readonly redisClient;

  constructor(serviceOptions: RedisClientOptions) {
    this.redisClient = createClient(serviceOptions);
```

```
  }

  getClient() {
    return this.redisClient;
  }

  onModuleInit() {
    this.redisClient.connect();
  }

  onModuleDestroy() {
    this.redisClient.disconnect();
  }
}
```

Файл template-service.ts

```
import * as fs from 'fs';
import { ZodObject } from 'zod';
import { ZodRawShape } from 'zod/lib/types';
import { compile } from 'handlebars';
import { Injectable, OnModuleInit } from '@nestjs/common';
import ITemplateService from '@v1/templates/interfaces/template-
service.interface';

@Injectable()
export default abstract class TemplateService<
  TSchema extends { [key: string]: unknown },
> implements OnModuleInit, ITemplateService<TSchema>
{
  protected delegate?: HandlebarsTemplateDelegate;

  protected constructor(
    protected readonly name: string,
    protected readonly schema: ZodObject<ZodRawShape>,
  ) {}

  public async onModuleInit() {
    await this.compile();
  }

  private async compile() {
```

```
    const template = await this.readTemplate();

    this.delegate = compile(template);
  }

  public async render(data: TSchema) {
    if (!this.delegate) {
      throw new Error(`Template "${this.name}" is not compiled`);
    }

    if (!this.validate(data)) {
      throw new Error(`Invalid data for template "${this.name}"`);
    }

    return this.delegate(data);
  }

  public validate(data: unknown): data is TSchema {
    const { success } = this.schema.safeParse(data);

    return success;
  }

  private async readTemplate() {
    return new Promise<string>((resolve, reject) => {
      fs.readFile(`./templates/${this.name}.hbs`, (err, data) => {
        if (err) reject(err);
        resolve(data.toString());
      });
    });
  }
}
```

Файл templates.provider.ts

```
import TemplateService from '@v1/templates/template-service';
import TemplateName from '@v1/templates/enums/template-name.enum';
import CookTemplateService from '@v1/templates/cook-template.service';
import { Injectable } from '@nestjs/common';

@Injectable()
export default class TemplatesProvider {
```

```
  constructor(private readonly cookTemplateService: CookTemplateService) {}

  getService<
    TSchema extends { [key: string]: unknown } = { [key: string]: unknown },
  >(templateName: string): TemplateService<TSchema> {
    switch (templateName) {
      case TemplateName.COOK:
        return this.cookTemplateService as unknown as
TemplateService<TSchema>;
      default:
        throw new Error('Invalid strategy');
    }
  }
}
```

Файл cook-template.service.ts

```
import { z } from 'zod';
import { Injectable } from '@nestjs/common';
import TemplateService from '@v1/templates/template-service';
import TemplateName from '@v1/templates/enums/template-name.enum';

export const cookTemplateSchema = z.object({
  'storage-endpoint': z.string().url(),
  header: z.object({
    logo: z.string(),
    'menu-item1': z.string(),
    'menu-item2': z.string(),
    'menu-item3': z.string(),
    'menu-item4': z.string(),
  }),
  greeting: z.object({
    title: z.string(),
    text: z.string(),
    btn: z.string(),
  }),
  advantages: z.object({
    title1: z.string(),
    text1: z.string(),
    title2: z.string(),
    text2: z.string(),
```

```
    title3: z.string(),
    text3: z.string(),
    title4: z.string(),
    text4: z.string(),
  }),
  buble: z.object({
    title: z.string(),
    text: z.string(),
    btn: z.string(),
  }),
  classes: z.object({
    title: z.string(),
    text: z.string(),
    btn: z.string(),
  }),
  mission: z.object({
    title: z.string(),
    'item-title1': z.string(),
    'item-text1': z.string(),
    'item-title2': z.string(),
    'item-text2': z.string(),
    'item-title3': z.string(),
    'item-text3': z.string(),
    'item-title4': z.string(),
    'item-text4': z.string(),
  }),
  programs: z.object({
    'item-number1': z.string(),
    'item-title1': z.string(),
    'item-text1': z.string(),
    'item-number2': z.string(),
    'item-title2': z.string(),
    'item-text2': z.string(),
    'item-number3': z.string(),
    'item-title3': z.string(),
    'item-text3': z.string(),
    'item-number4': z.string(),
    'item-title4': z.string(),
    'item-text4': z.string(),
    'item-number5': z.string(),
    'item-title5': z.string(),
    'item-text5': z.string(),
    'item-number6': z.string(),
```

```
    'item-title6': z.string(),
    'item-text6': z.string(),
  }),
  description: z.object({
    'left-title': z.string(),
    'right-title': z.string(),
    'right-text1': z.string(),
    'right-text2': z.string(),
    btn: z.string(),
  }),
  footer: z.object({
    info1: z.string(),
    info2: z.string(),
    info3: z.string(),
    form: z.object({
      title: z.string(),
      text: z.string(),
    }),
  }),
});

export type CookTemplateData = z.infer<typeof cookTemplateSchema>;

@Injectable()
export default class CookTemplateService extends
TemplateService<CookTemplateData> {
  constructor() {
    super(TemplateName.COOK, cookTemplateSchema);
  }
}
```

Файл user-templates.controller.ts

```
import { Request } from 'express';
import {
  Req,
  Get,
  Post,
  Body,
  Param,
  Patch,
  Inject,
```

```
  Header,
  Delete,
  HttpCode,
  UseGuards,
  Controller,
  ParseIntPipe,
  BadRequestException,
  HttpStatus,
} from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { ConfigType } from '@nestjs/config';
import { ApiBearerAuth, ApiTags } from '@nestjs/swagger';
import AuthUser from '@v1/auth/decorators/auth-user.decorator';
import { IAccessPayload } from '@v1/auth/interface/access-payload.interface';
import UserTemplatesService from '@v1/user-templates/user-templates.service';
import CreateTemplateDto from '@v1/user-templates/dto/create-template.dto';
import TemplatesConstants from '@v1/templates/templates.constants';
import TemplatesProvider from '@v1/templates/templates.provider';
import JwtAccessGuard from '@v1/auth/guards/jwt-access.guard';
import UpdateTemplateDto from '@v1/user-templates/dto/update-template.dto';
import {
  awsNamespaceKey,
  awsNamespace,
} from 'src/modules/app/config/aws.config';
import {
  jwtNamespaceKey,
  jwtNamespace,
} from 'src/modules/app/config/jwt.config';

@ApiTags('User Templates')
@ApiBearerAuth()
@Controller({ path: 'templates', version: '1' })
export default class UserTemplatesController {
  constructor(
    @Inject(awsNamespaceKey)
    private readonly awsConfig: ConfigType<awsNamespace>,
    private readonly usersTemplatesService: UserTemplatesService,
    private readonly templatesProvider: TemplatesProvider,
    private readonly jwtService: JwtService,
    @Inject(jwtNamespaceKey) private readonly secrets:
ConfigType<jwtNamespace>,
  ) {}
```

```
@Post()
@UseGuards(JwtAccessGuard)
async create(
  @AuthUser() user: IAccessPayload,
  @Body() { name }: CreateTemplateDto,
) {
  const { templatesData } = TemplatesConstants;

  return this.usersTemplatesService.create({
    name,
    data: templatesData[name],
    userId: user.id,
  });
}

@Get(':id')
@UseGuards(JwtAccessGuard)
async getOne(
  @Param('id', ParseIntPipe) id: number,
  @AuthUser() user: IAccessPayload,
) {
  const template = await this.usersTemplatesService.findOneOrError({
    userId: user.id,
    id,
  });

  return {
    data: template,
  };
}

@Get('link/:id')
@UseGuards(JwtAccessGuard)
async getLink(@Req() req: Request, @Param('id', ParseIntPipe) id: number) {
  const url = `${req.protocol}://${req.headers.host}`;
  const token = this.jwtService.sign(
    { id },
    { secret: this.secrets.accessSecret },
  );

  return {
    data: `${url}/api/v1/templates/render/${token}`,
  };
```

```
  }

  @Get('render/:token')
  @Header('Content-Type', 'text/html')
  async renderTemplate(@Param('token') token: string) {
    const { id } = this.jwtService.verify(token, {
      secret: this.secrets.accessSecret,
    });
    const { data, name } = await this.usersTemplatesService.findOneOrError({
      id,
    });
    const { endpoint } = this.awsConfig;

    return this.templatesProvider.getService(name).render({
      ...data,
      'storage-endpoint': `${endpoint}`,
    });
  }

  @Get()
  @UseGuards(JwtAccessGuard)
  async getTemplates(@AuthUser() user: IAccessPayload) {
    const templates = await this.usersTemplatesService.find({
      userId: user.id,
    });

    return {
      data: templates,
    };
  }

  @Patch(':id')
  @UseGuards(JwtAccessGuard)
  async update(
    @AuthUser() user: IAccessPayload,
    @Param('id', ParseIntPipe) id: number,
    @Body() { data }: UpdateTemplateDto,
  ) {
    const { endpoint } = this.awsConfig;
    const { name } = await this.usersTemplatesService.findOneOrError({
      userId: user.id,
      id,
    });
```

```
    const updateData = {
      'storage-endpoint': `${endpoint}`,
      ...data,
    };
    const isValid = this.templatesProvider
      .getService(name)
      .validate(updateData);

    if (!isValid) {
      throw new BadRequestException(`Invalid data for template "${name}"`);
    }

    return this.usersTemplatesService.update(
      { userId: user.id, id },
      {
        data: updateData,
      },
    );
  }

  @Delete(':id')
  @HttpCode(HttpStatus.NO_CONTENT)
  @UseGuards(JwtAccessGuard)
  async delete(
    @AuthUser() user: IAccessPayload,
    @Param('id', ParseIntPipe) id: number,
  ) {
    const { affected } = await this.usersTemplatesService.delete({
      userId: user.id,
      id,
    });

    if (!affected || affected === 0) {
      throw new BadRequestException('Template not found');
    }
  }
}
```

Файл user-templates.repository.ts

```
import { Repository } from 'typeorm';
import { Injectable } from '@nestjs/common';
```

```
import { InjectRepository } from '@nestjs/typeorm';
import UserTemplateEntity from '@v1/user-templates/schemas/user-
template.entity';
import {
  CreateTemplate,
  FindTemplate,
  UpdateTemplate,
} from '@v1/user-templates/user-templates';

@Injectable()
export default class UserTemplatesRepository {
  constructor(
    @InjectRepository(UserTemplateEntity)
    private readonly userTemplatesModel: Repository<UserTemplateEntity>,
  ) {}

  async create({
    userId,
    ...payload
  }: CreateTemplate): Promise<UserTemplateEntity> {
    return this.userTemplatesModel.save({
      ...payload,
      user: {
        id: userId,
      },
    });
  }

  async findOneOrError(where: FindTemplate): Promise<UserTemplateEntity> {
    return this.userTemplatesModel.findOneOrFail({ where });
  }

  async find(where: FindTemplate): Promise<UserTemplateEntity[]> {
    return this.userTemplatesModel.find({ where });
  }

  async update(where: FindTemplate, payload: UpdateTemplate) {
    return this.userTemplatesModel.update(where, payload);
  }

  async delete(where: FindTemplate) {
    return this.userTemplatesModel.delete(where);
  }
```

```
}
```

Файл user-templates.service.ts

```
import { Injectable } from '@nestjs/common';
import UserTemplatesRepository from '@v1/user-templates/user-
templates.repository';
import {
  CreateTemplate,
  FindTemplate,
  UpdateTemplate,
} from '@v1/user-templates/user-templates';
import S3ManagerService from '@v1/aws/s3-manager.service';

@Injectable()
export default class UserTemplatesService {
  constructor(
    private readonly userTemplatesRepository: UserTemplatesRepository,
    private readonly s3: S3ManagerService,
  ) {}

  async create(payload: CreateTemplate) {
    return this.userTemplatesRepository.create(payload);
  }

  async findOneOrError(where: FindTemplate) {
    return this.userTemplatesRepository.findOneOrError(where);
  }

  async find(where: FindTemplate) {
    return this.userTemplatesRepository.find(where);
  }

  async update(where: FindTemplate, payload: UpdateTemplate) {
    return this.userTemplatesRepository.update(where, payload);
  }

  async delete(where: FindTemplate) {
    return this.userTemplatesRepository.delete(where);
  }
}
```

Файл users.controller.ts

```
import {
  Body,
  Controller,
  Get,
  Param,
  ParseIntPipe,
  Patch,
  UseGuards,
} from '@nestjs/common';
import {
  ApiBearerAuth,
  ApiExtraModels,
  ApiOkResponse,
  ApiTags,
} from '@nestjs/swagger';

import Serialize from '@decorators/serialization.decorator';
import UsersService from '@v1/users/users.service';
import UserEntity from '@v1/users/schemas/user.entity';
import JwtAccessGuard from '@v1/auth/guards/jwt-access.guard';
import AuthUser from '@v1/auth/decorators/auth-user.decorator';
import { IAccessPayload } from '@v1/auth/interface/access-payload.interface';
import UpdateUserDto from '@v1/users/dto/update/update-user.dto';

@ApiTags('Users')
@ApiBearerAuth()
@ApiExtraModels(UserEntity)
@Controller({ version: '1', path: 'users' })
export default class UsersController {
  constructor(private readonly usersService: UsersService) {}

  @ApiOkResponse()
  @UseGuards(JwtAccessGuard)
  @Serialize(UserEntity)
  @Get(':id')
  async getOne(@Param('id', ParseIntPipe) id: number): Promise<UserEntity> {
    return this.usersService.getOneOrError({ id });
  }

  @ApiOkResponse()
```

```
  @UseGuards(JwtAccessGuard)
  @Serialize(UserEntity)
  @Patch('self')
  async updateSelf(
    @AuthUser() authUser: IAccessPayload,
    @Body() attributes: UpdateUserDto,
  ): Promise<UserEntity> {
    await this.usersService.update({ id: authUser.id }, attributes);

    return this.usersService.getOneOrError({ id: authUser.id });
  }
}
```

Файл auth.controller.ts

```
import {
  ApiBearerAuth,
  ApiBody,
  ApiCreatedResponse,
  ApiExtraModels,
  ApiOkResponse,
  ApiTags,
  ApiUnauthorizedResponse,
} from '@nestjs/swagger';
import {
  Body,
  Controller,
  HttpCode,
  HttpStatus,
  Post,
  UseGuards,
} from '@nestjs/common';

import ApiResponses from '@decorators/typical-response.decorator';
import Serialize from '@decorators/serialization.decorator';
import AuthService, { JwtTokens } from '@v1/auth/auth.service';
import AuthUser from '@v1/auth/decorators/auth-user.decorator';
import LocalAuthGuard from '@v1/auth/guards/local-auth.guard';
import JwtRefreshGuard from '@v1/auth/guards/jwt-refresh.guard';
import UserEntity from '@v1/users/schemas/user.entity';
import CreateUserDto from '@v1/users/dto/create/create-user.dto';
import SignInDto from '@v1/auth/dto/sign-in.dto';
```

```
@ApiTags('Auth')
@Controller({ version: '1' })
@ApiExtraModels(UserEntity)
export default class AuthController {
  constructor(private readonly authService: AuthService) {}

  @ApiBody({ type: SignInDto })
  @ApiOkResponse()
  @ApiResponses()
  @HttpCode(HttpStatus.OK)
  @UseGuards(LocalAuthGuard)
  @Serialize(UserEntity)
  @Post('sign-in')
  async signIn(
    @AuthUser() validatedUser: UserEntity,
  ): Promise<{ tokens: JwtTokens; user: UserEntity }> {
    const jwtTokens = await this.authService.addSession(validatedUser);

    return {
      tokens: jwtTokens,
      user: validatedUser,
    };
  }

  @ApiCreatedResponse()
  @ApiResponses()
  @HttpCode(HttpStatus.CREATED)
  @Serialize(UserEntity)
  @Post('sign-up')
  async signUp(@Body() attributes: CreateUserDto): Promise<UserEntity> {
    return this.authService.registerUser(attributes);
  }

  @ApiOkResponse()
  @ApiResponses()
  @ApiUnauthorizedResponse({
    description: '401. Unauthorized',
  })
  @ApiBearerAuth()
  @UseGuards(JwtRefreshGuard)
  @HttpCode(HttpStatus.OK)
  @Serialize(UserEntity)
```

```
  @Post('sign-in-refresh')
  async signInByRefresh(
    @AuthUser() validatedUser: UserEntity,
  ): Promise<{ tokens: JwtTokens; user: UserEntity }> {
    const jwtTokens = await this.authService.addSession(validatedUser);

    return {
      tokens: jwtTokens,
      user: validatedUser,
    };
  }
}
```

Файл app.filter.ts

```
import { Response } from 'express';

import { HttpStatus } from '@nestjs/common';
import {
  ArgumentsHost,
  ExceptionFilter,
  HttpArgumentsHost,
  WsArgumentsHost,
} from '@nestjs/common/interfaces';

abstract class AppFilter<Exception = unknown> implements ExceptionFilter {
  catch(exception: Exception, host: ArgumentsHost) {
    switch (host.getType()) {
      case 'ws':
        return this.catchWsException(exception, host.switchToWs());
      case 'http':
      default:
        return this.catchHttpException(exception, host.switchToHttp());
    }
  }

  catchHttpException(exception: Exception, host: HttpArgumentsHost) {
    const res = host.getResponse<Response>();
    const statusCode = this.httpStatus(exception);

    return res
      .status(statusCode)
```

```
        .json(this.errorAttributes(exception, statusCode));
    }

    catchWsException(exception: Exception, host: WsArgumentsHost) {
      const client = host.getClient();
      const statusCode = this.wsStatus(exception);

      return client.emit('error', this.errorAttributes(exception, statusCode));
    }

    abstract httpStatus(exception: Exception): HttpStatus;

    abstract wsStatus(exception: Exception): string | number;

    abstract errorAttributes(
      exception: Exception,
      statusCode: number | string,
    ): unknown;
}

export default AppFilter;
```

Файл all-exceptions.filter.ts

```
import { Catch, HttpException, HttpStatus, Logger } from '@nestjs/common';

import AppFilter from '@filters/app.filter';

@Catch()
export default class AllExceptionsFilter extends AppFilter {
  private readonly logger: Logger = new Logger(AllExceptionsFilter.name);

  errorAttributes(exception: unknown, statusCode: number | string) {
    if (exception instanceof HttpException) {
      return this.knownErrorAttributes(exception, statusCode);
    }
    this.logger.error(exception);

    return this.unknownErrorAttributes(statusCode);
  }

  private unknownErrorAttributes(statusCode: number | string) {
```

```
    return {
      title: 'Unknown error',
      code: '1',
      status: `${statusCode}`,
      message: 'Something went wrong',
    };
  }

  private knownErrorAttributes(
    exception: HttpException,
    statusCode: number | string,
  ) {
    return {
      title: exception.name,
      code: '1',
      status: `${statusCode}`,
      message: exception.message,
    };
  }

  httpStatus(exception: unknown) {
    return exception instanceof HttpException
      ? exception.getStatus()
      : HttpStatus.INTERNAL_SERVER_ERROR;
  }

  wsStatus() {
    return '1011';
  }
}
```

Файл bad-request-exceptions.filter.ts

```
import { Catch, BadRequestException, HttpStatus } from '@nestjs/common';
import AppFilter from '@filters/app.filter';
import { Constraints } from '@utils/transform-validation-errors';

@Catch(BadRequestException)
export default class BadRequestExceptionFilter extends
AppFilter<BadRequestException> {
  isValidationErrorResponse(response: object): response is {
    error: 'Validation error';
```

```
    message: Constraints[];
  } {
    return (
      'error' in response &&
      response.error === 'Validation error' &&
      'message' in response &&
      Array.isArray(response.message)
    );
  }

  isRegularErrorResponse(response: object): response is {
    error: string;
    message: string;
  } {
    return (
      'error' in response &&
      typeof response.error === 'string' &&
      'message' in response &&
      typeof response.message === 'string'
    );
  }

  errorAttributes(exception: BadRequestException, statusCode: number) {
    const response = exception.getResponse();
    let detail: string;
    let errorsMeta: (string | Constraints)[];

    if (typeof response === 'string') {
      detail = response;
      errorsMeta = [response];
    } else if (this.isValidationErrorResponse(response)) {
      detail = `Some fields did not pass validation: ${response.message
        .map(({ constraints }) => constraints.join(', '))
        .join(', ')}`;
      errorsMeta = response.message;
    } else if (this.isRegularErrorResponse(response)) {
      detail = response.message;
      errorsMeta = [response.message];
    } else {
      detail = 'Bad request';
      errorsMeta = ['Unknown error response type'];
    }
```

```
    return {
      title: 'ValidationError',
      detail,
      code: '42000',
      status: `${statusCode}`,
      meta: {
        errors: errorsMeta,
      },
    };
  }

  httpStatus(): HttpStatus {
    return HttpStatus.BAD_REQUEST;
  }

  wsStatus(): string | number {
    return HttpStatus.BAD_REQUEST;
  }
}
```

Файл entity-not-found.filter.ts

```
import { EntityNotFoundError } from 'typeorm';

import { Catch, HttpStatus } from '@nestjs/common';
import AppFilter from '@filters/app.filter';

@Catch(EntityNotFoundError)
export default class EntityNotFoundFilter extends
AppFilter<EntityNotFoundError> {
  private readonly entityRegex = /entity of type "(.*?)"/g;

  private retrieveEntityName(message: string): string {
    return this.entityRegex.exec(message)?.[1] ?? 'UnknownEntity';
  }

  errorAttributes(exception: EntityNotFoundError, statusCode: number |
string) {
    const entityName = this.retrieveEntityName(exception.message);

    return {
      title: exception.name,
```

```
      detail: `${entityName} not found`,
      code: '02000',
      status: `${statusCode}`,
      links: {
        type:
'https://typeorm.delightful.studio/classes/_error_entitynotfounderror_.entity
notfounderror.html',
      },
    };
  }

  httpStatus() {
    return HttpStatus.NOT_FOUND;
  }

  wsStatus() {
    return HttpStatus.NOT_FOUND;
  }
}
```

Файл unauthorized-exceptions.filter.ts

```
import { Catch, HttpStatus, UnauthorizedException } from '@nestjs/common';
import AppFilter from '@filters/app.filter';

@Catch(UnauthorizedException)
export default class UnauthorizedExceptionsFilter extends
AppFilter<UnauthorizedException> {
  errorAttributes(exception: UnauthorizedException, statusCode: number) {
    return {
      title: 'UnauthorizedError',
      detail: 'Your token is invalid or expired',
      code: '4',
      status: `${statusCode}`,
    };
  }

  httpStatus() {
    return HttpStatus.UNAUTHORIZED;
  }

  wsStatus() {
```

```
    return HttpStatus.UNAUTHORIZED;
  }
}
```

Файл cook.hbs

```
<html>
<body>
    <div class="wrapper">
        <header class="header">
            <div class="header__wrap-logo">
                <a href="#logo" class="header__logo">{{header.logo}}</a>
            </div>
            <nav class="header__nav">
                <a href="#advantages">{{header.menu-item1}}</a>
                <a href="#mission">{{header.menu-item2}}</a>
                <a href="#why_us">{{header.menu-item3}}</a>
                <a href="#contact_us">{{header.menu-item4}}</a>
            </nav>
        </header>
        <div class="greeting">
            <div class="greeting__container">
                <div class="greeting__text">
                    <h2>{{greeting.title}}</h2>
                    <span></span>
                    <p>{{greeting.text}}</p>
                    <button id="greeting__btn">{{greeting.btn}}</button>
                </div>
                <div class="greeting__img">
                    <img src="{{storage-endpoint}}/cook.png" alt="cook">
                </div>
            </div>
        </div>
        <div class="advantages" id="advantages">
            <div class="advantages__container">
                <div class="advantages__item">
                    <span id="first__icon"></span>
                    <div class="advantages__text">
                        <h4>{{advantages.title1}}</h4>
                        <p>{{advantages.text1}}</p>
                    </div>
                </div>
```

```
                <div class="advantages__item">
                    <span id="second__icon"></span>
                    <div class="advantages__text">
                        <h4>{{advantages.title2}}</h4>
                        <p>{{advantages.text2}}</p>
                    </div>
                </div>
                <div class="advantages__item">
                    <span id="third__icon"></span>
                    <div class="advantages__text">
                        <h4>{{advantages.title3}}</h4>
                        <p>{{advantages.text3}}</p>
                    </div>
                </div>
                <div class="advantages__item">
                    <span id="fourth__icon"></span>
                    <div class="advantages__text">
                        <h4>{{advantages.title4}}</h4>
                        <p>{{advantages.text4}}</p>
                    </div>
                </div>
            </div>
        </div>
        <div class="buble">
            <div class="buble__container">
                <div class="buble__bagel">
                    <span id="yellow_big"></span>
                    <span id="white_big"></span>
                    <span id="yellow_small"></span>
                    <span id="white_small"></span>
                </div>
                <div class="buble__img">
                    <span id="yellow__img"></span>
                    <img src="{{storage-endpoint}}/buble_cook.jpg" alt="">
                </div>
                <div class="buble_big">
                    <span>
                        <h1>{{buble.title}}</h1>
                        <p>{{buble.text}}</p>
                        <button id="buble__btn">{{buble.btn}}</button>
                    </span>
                </div>
                <div class="buble_small">
```

```
                    <span></span>
                </div>
            </div>
        </div>
        <div class="classes">
            <div class="classes__container">
                <img id="classes__ck1" src="{{storage-
endpoint}}/class_cook1.jpg" alt="cook1"></img>
                <img id="classes__ck2" src="{{storage-
endpoint}}/class_cook2.jpg" alt="cook2"></img>
                <img id="classes__ck3" src="{{storage-
endpoint}}/class_cook3.jpg" alt="cook3"></img>
                <img id="classes__ck4" src="{{storage-endpoint}}/food.jpg"
alt="food"></img>
                <div class="classes__text">
                    <h2>{{classes.title}}</h2>
                    <p>{{classes.text}}</p>
                    <button id="classes__btn">{{classes.btn}}</button>
                </div>
            </div>
        </div>
        <div class="mission" id="mission">
            <div class="mission__title" >
                <h2>{{mission.title}}</h2>
            </div>
            <div class="mission__container">
                <div class="mission__item">
                    <span></span>
                    <h4>{{mission.item-title1}}</h4>
                    <p>{{mission.item-text1}}</p>
                </div>
                <div class="mission__item">
                    <span></span>
                    <h4>{{mission.item-title2}}</h4>
                    <p>{{mission.item-text2}}</p>
                </div>
                <div class="mission__item">
                    <span></span>
                    <h4>{{mission.item-title3}}</h4>
                    <p>{{mission.item-text3}}</p>
                </div>
                <div class="mission__item">
                    <span></span>
```

```
                <h4>{{mission.item-title4}}</h4>
                <p>{{mission.item-text4}}</p>
            </div>
        </div>
    </div>
    <div class="background"></div>
    <div class="program">
        <div class="program__container">
            <div class="program__item">
                <span>
                    <h4>01</h4>
                </span>
                <h4>{{programs.item-title1}}</h4>
                <p>{{programs.item-text1}}</p>
            </div>
            <div class="program__item">
                <span>
                    <h4>02</h4>
                </span>
                <h4>{{programs.item-title2}}</h4>
                <p>{{programs.item-text2}}</p>
            </div>
            <div class="program__item">
                <span>
                    <h4>03</h4>
                </span>
                <h4>{{programs.item-title3}}</h4>
                <p>{{programs.item-text3}}</p>
            </div>
            <div class="program__item">
                <span>
                    <h4>04</h4>
                </span>
                <h4>{{programs.item-title4}}</h4>
                <p>{{programs.item-text4}}</p>
            </div>
            <div class="program__item">
                <span>
                    <h4>05</h4>
                </span>
                <h4>{{programs.item-title5}}</h4>
                <p>{{programs.item-text5}}</p>
            </div>
```

```
                <div class="program__item">
                    <span>
                        <h4>06</h4>
                    </span>
                    <h4>{{programs.item-title6}}</h4>
                    <p>{{programs.item-text6}}</p>
                </div>
            </div>
        </div>
        <div class="description" id="why_us">
            <div class="description__container">
                <div class="description__left">
                    <h5>{{description.left-title}}</h5>
                    <span></span>
                    <img src="{{storage-endpoint}}/description.jpg"
alt="description_img">
                </div>
                <div class="description__right">
                    <h2>{{description.right-title}}</h2>
                    <p>{{description.right-text1}}</p>
                    <p>{{description.right-text2}}</p>
                    <button
id="description__btn">{{description.btn}}</button>
                </div>
            </div>
        </div>
        <footer class="footer" id="contact_us">
            <div class="footer__container">
                <div class="footer__info">
                    <div class="footer__info-item">
                        <img src="{{storage-endpoint}}/location_icon.png"
alt="location_icon">
                        <h5>LOCATION</h5>
                        <p>{{footer.info1}}</p>
                    </div>
                    <div class="footer__info-item">
                        <img src="{{storage-endpoint}}/phone_icon.png"
alt="phone_icon">
                        <h5>CALL US</h5>
                        <p>{{footer.info2}}</p>
                    </div>
                    <div class="footer__info-item">
```

```
                        <img src="{{storage-endpoint}}/time_icon.png"
alt="time_icon">
                        <h5>BUSINESS HOURS</h5>
                        <p>{{footer.info3}}</p>
                    </div>
                </div>
                <div class="footer__contact">
                    <h3>Contact Us</h3>
                    <p>{{footer.form.text}}</p>
                    <input type="text" placeholder="Enter your Name">
                    <input type="email" name="" id="" placeholder="Enter a
valid email address">
                    <textarea name="" id="" cols="30" rows="10"></textarea>
                    <button id="footer__btn">Submit</button>
                </div>
                <div class="footer__img">
                    <img src="{{storage-endpoint}}/footer.jpg"
alt="footer_img">
                </div>
            </div>
        </footer>
    </div>
</body>
</html>
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

ВЕБ-СЕРВІС ДЛЯ ГЕНЕРАЦІЇ ТА РОЗГОРТАННЯ ЛЕНДИНГІВ

Програма та методика тестування

КПІ. ІТ-9314.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Микола ХРАМЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Станіслав КЛЕПІКОВ

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є веб-сервіс для генерації та розгортання лендингів, що являє собою клієнт-серверним веб-застосунком та розроблене під різні операційні системи з підтримкою усіх сучасних браузерів.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

перевірка сумісності веб-додатку з останніми версіями сучасних браузерів таких як: Google Chrome, Mozilla Firefox;

перевірка сумісності застосунку з різними операційними системами: Windows, Linux, MacOS;

знаходження проблем, помилок і недоліків з метою їх усунення;

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;

мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- тестування на виведення повідомлень про помилку, коли це необхідно;
- тестування зміни розміру екрану;
- тестування інтерфейсу користувача;
- тестування зручності використання;

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

ПРОГРАМНИЙ ЗАСТОСУНОК ДЛЯ КОРПОРАТИВНОГО НАВЧАННЯ

Керівництво користувача

КПІ.ІТ-9314.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Микола ХРАМЧЕНКО

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Станіслав КЛЕПІКОВ

Київ – 2023

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Розробка веб-сервісу для генерації та розгортання лендингів призначена для спрощення та прискорення процесу створення та розгортання лендингів з метою вирішення проблеми складності і затрат, пов'язаних з цими процесами та може використовуватись підприємцями та власниками бізнесу.

Кінцева збірка програмного забезпечення включає готову до використання версію системи, яка включає всі необхідні компоненти та налаштування. Ця збірка містить всі необхідні файли та ресурси, необхідні для належної роботи системи.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– наявність доступу до Інтернету;

– наявність сучасного браузера, такого як Google Chrome або Mozilla Firefox.

− тип процесору: Intel Core i3;

− об‘єм ОЗП: 4 Гб;

2.2 Завантаження застосунку

Для отримання доступу до ПЗ в будь-якому браузері потрібно відкрити адресу сторінки сайту.

2.3 Перевірка коректної роботи

При успішному завантаженні застосунку повинна відображатись головна сторінка з опціями авторизації та реєстрації у верхньому меню.

3 ВИКОНАННЯ ПРОГРАМИ

Реєстрація:

Для початку роботи на сторінці реєстрації (рисунок 3.1) користувач заповнює обов'язкові поля, такі як ім'я, прізвище, електронна пошта та пароль. Після введення всіх необхідних даних користувач натискає кнопку "Зареєструватися".

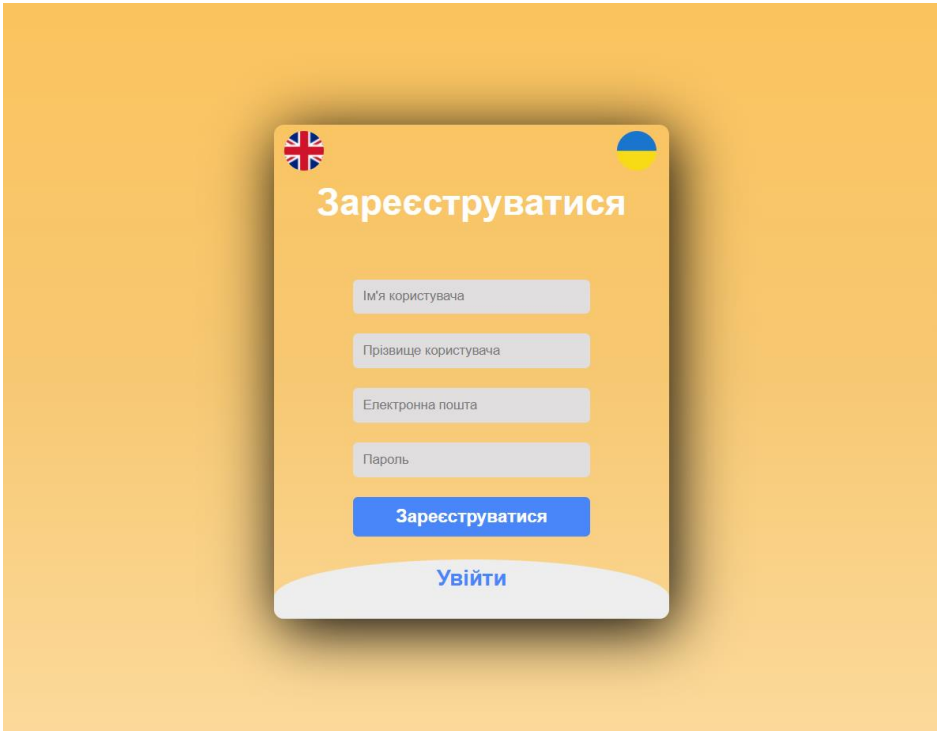


Рисунок 3.1 – Сторінка реєстрації

Після успішної реєстрації користувач може використовувати свій обліковий запис для входу в систему.

Авторизація:

Процес авторизації передбачає підтвердження ідентичності користувача для отримання доступу до системи. На сторінці авторизації (рисунок 3.2) користувач вводить свої облікові дані, такі як електронна пошта і пароль.

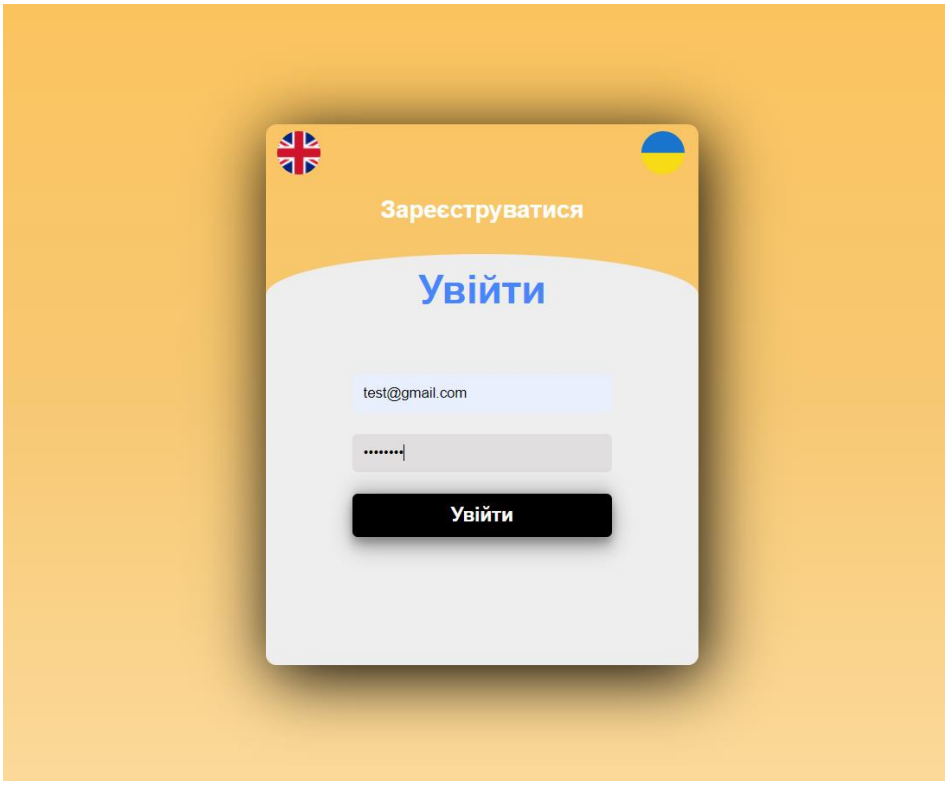


Рисунок 3.2 – Сторінка авторизації

Після введення облікових даних, програмне забезпечення перевіряє їх на відповідність зареєстрованим обліковим записам. Після успішної авторизації користувач отримує повний доступ до функціоналу програмного забезпечення, пов'язаного з його обліковим записом.

Також користувач має змогу вийти з особистого акаунту (рисунок 3.3).

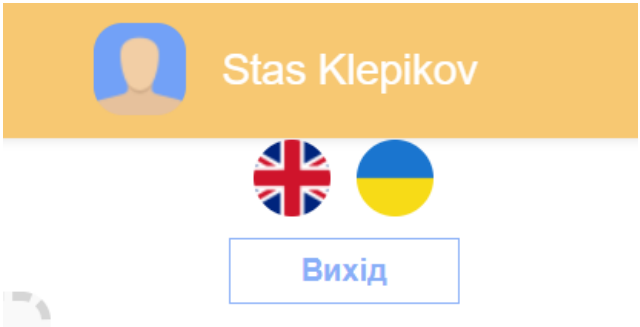


Рисунок 3.3 – Вихід з акаунту користувача

Створення веб-сторінки на основі шаблону:

Користувач може створити нову веб-сторінку натиснувши кнопку «+» (рисунок 3.4) та вибрати шаблон зі списку шаблонів в модальному вікні (рисунок 3.5).

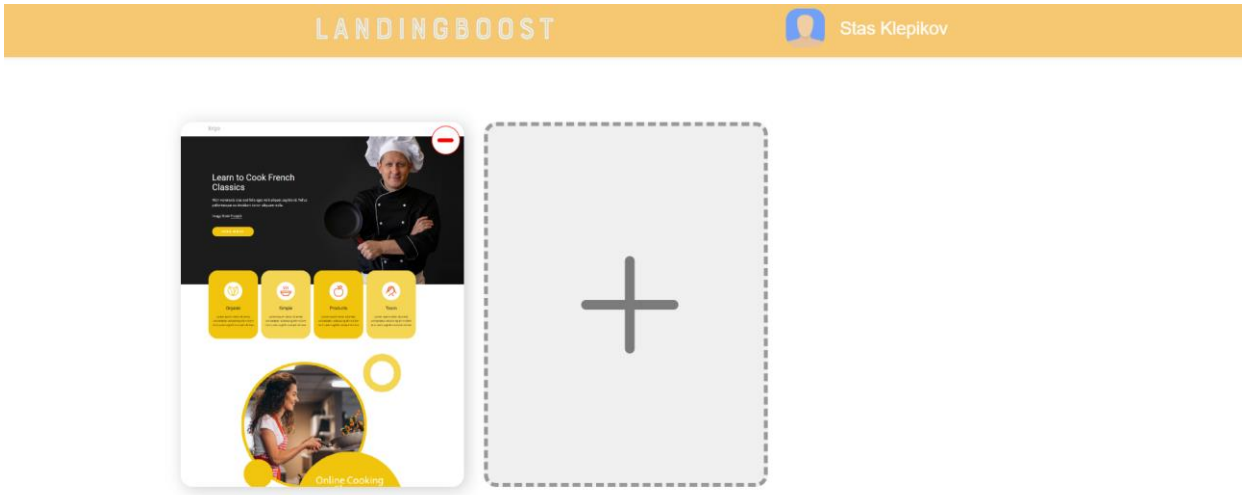


Рисунок 3.4 – Домашня сторінка профілю

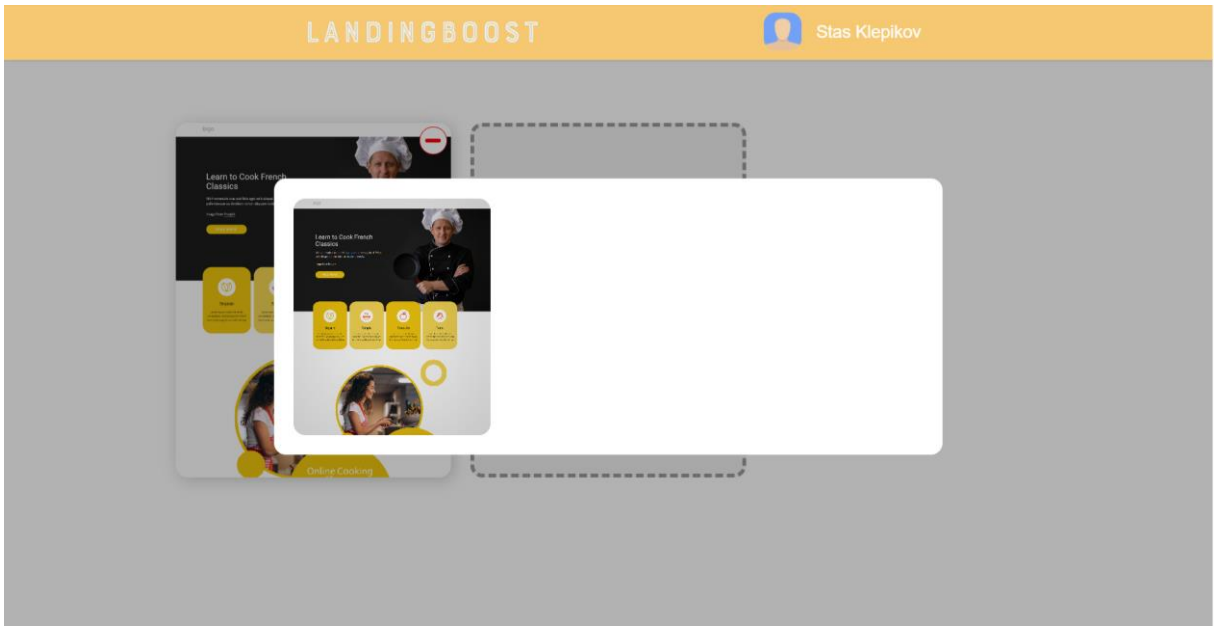


Рисунок 3.5 – Список шаблонів в модальному вікні

Перегляд створенних веб-сторінок:

Після створення нової веб-сторінки, повинна створитися картка цього шаблона на домашній сторінці, де користувач може переглянути коротку інформацію про створену веб-сторінку, а також редагувати її або розгорнути (рисунок 3.6).

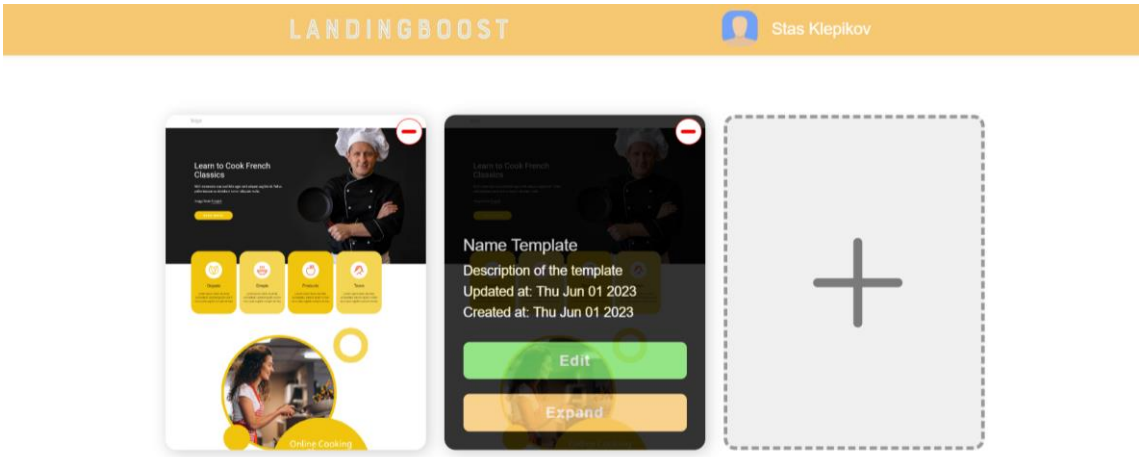


Рисунок 3.6 – Коротка інформація про шаблон, а також кнопки «Редагувати» та «Розгорнути»

Редагування веб-сторінки:

При натисканні кнопки «Редагувати» користувача перенаправляє на веб-сторінку шаблону в режимі редагування, де він може змінювати наповнення веб-сторінки та кольорову гаму. Відредагувавши шаблон, користувач натискає кнопку «Зберегти», щоб зберегти зміни в шаблоні (рисунок 3.7);

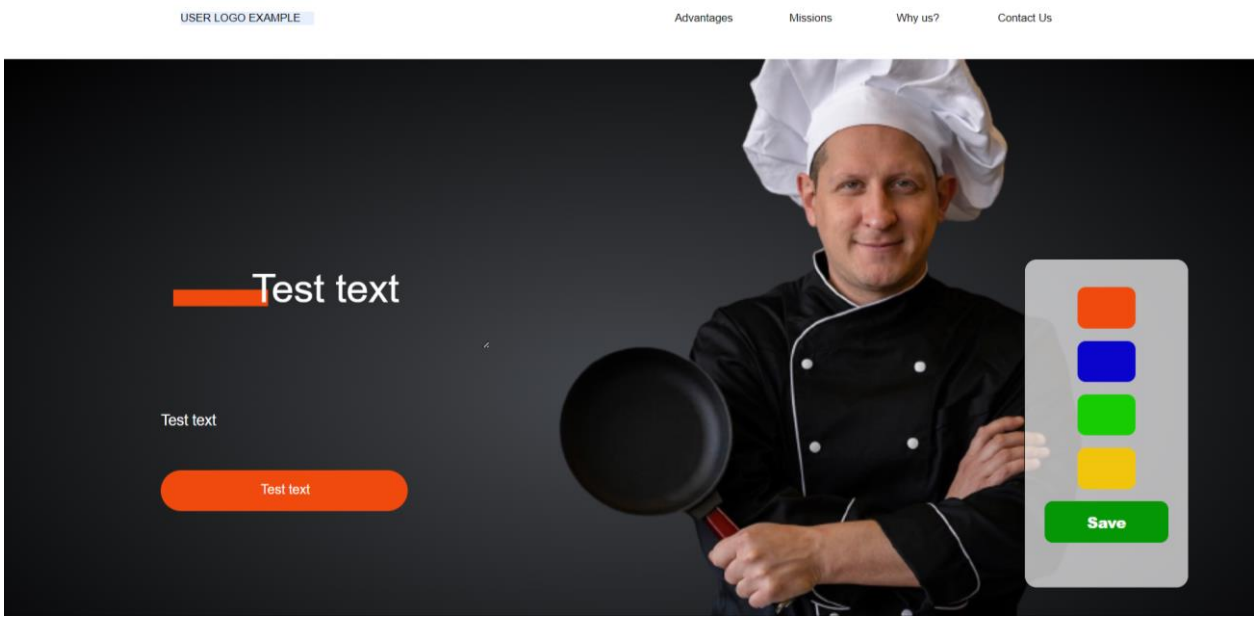


Рисунок 3.7 – Веб-сторінка в режимі редагування

Розгортання веб-сторінки:

Після внесення та збереження змін в шаблоні, користувач може розгорнути веб-сторінку, натиснувши кнопку «Розгорнути», та подивитися на результати редагування (рисунок 3.8);

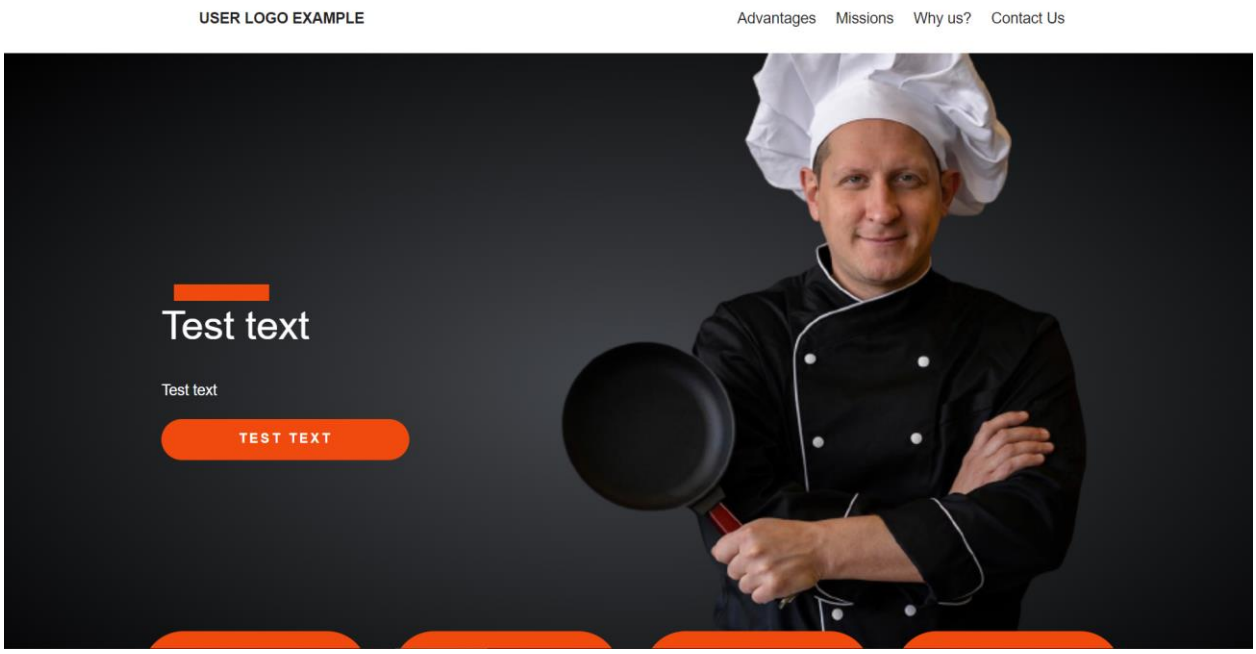


Рисунок 3.8 – Розгорнута веб-сторінка з внесеними змінами

Видалення веб-сторінки:

Якщо користувачу потрібно видалити веб-сторінку, він може натиснути кнопку «-» в верхньому правому куті та веб-сторінка зникне з домашньої сторінки користувача та видалиться (Рисунок 3.9);

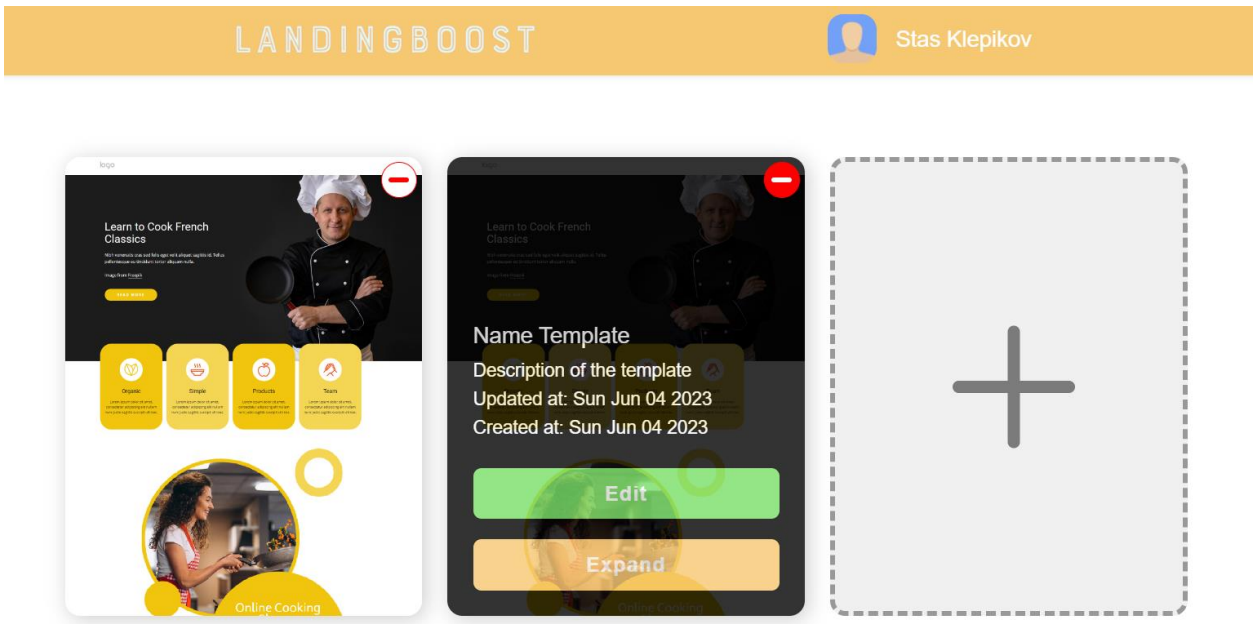


Рисунок 3.9 – Видалення веб-сторінки

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

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

ПРОГРАМНИЙ ЗАСТОСУНОК ДЛЯ КОРПОРАТИВНОГО НАВЧАННЯ

Графічний матеріал

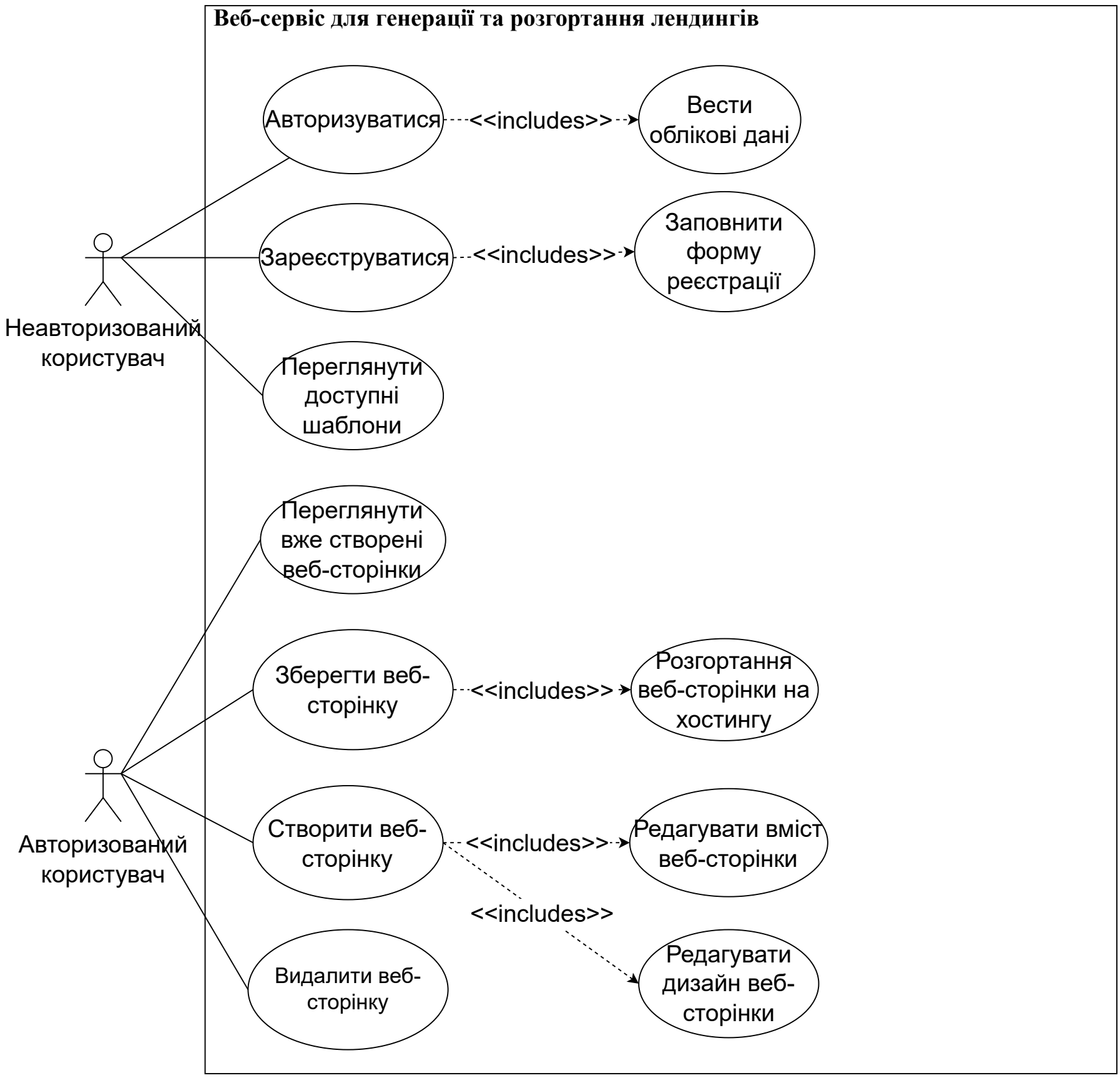
КПІ.ІТ-9314.045440.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Микола ХРАМЧЕНКО

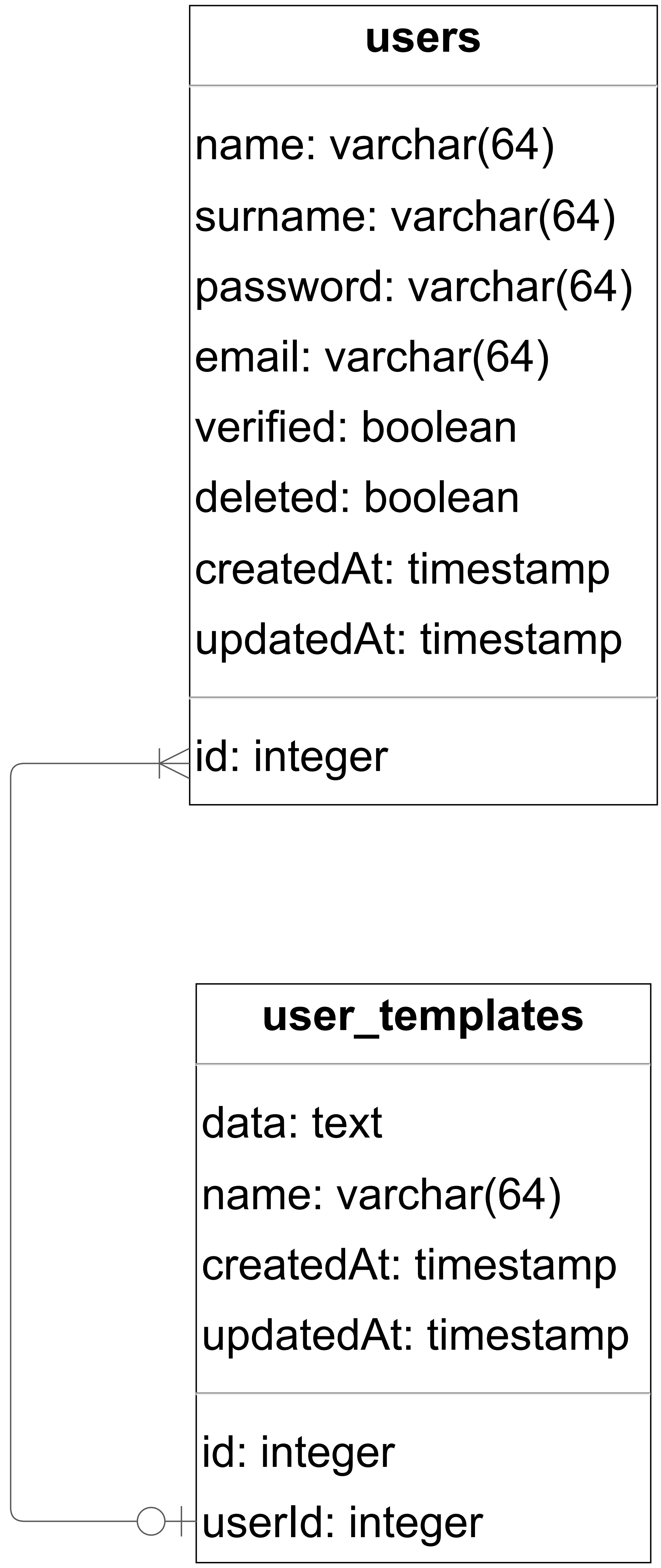
Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Станіслав КЛЕПІКОВ

Київ – 2023



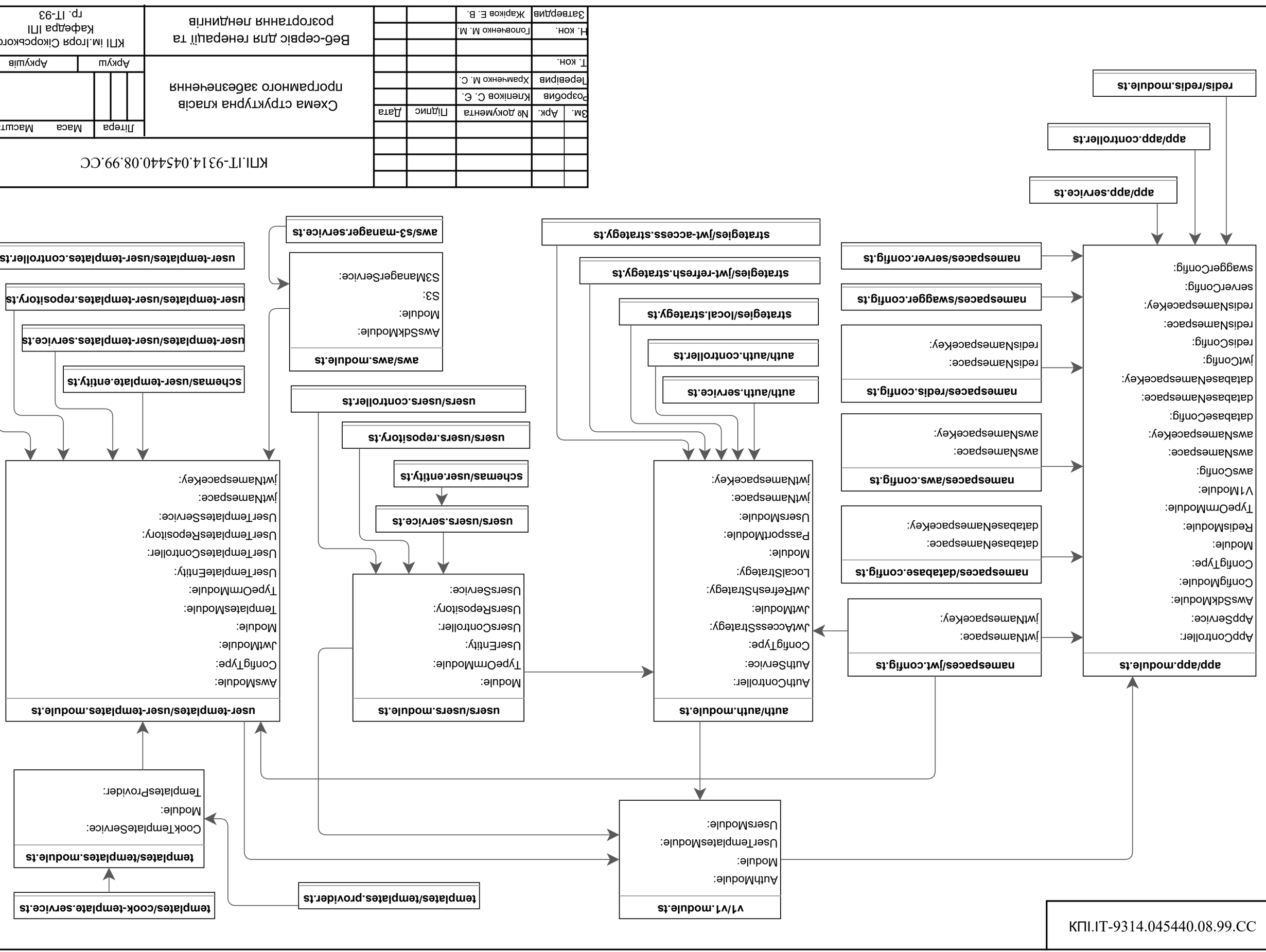
					КПІ.ІТ-9314.045440.06.99.СС	
					Схема структурна варіантів використань	Лит. / Арк. / Аркушів: 1
Зм.	Арк.	№ докум.	Підп.	Дата		Аркуш / Аркушів
Розроб.		Клепіков С. Є.			Веб-сервіс для генерації та розгортання лендингів	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-93
Перев.		Храмченко М. С.				
Т. Кон.						
Н. Кон.		Головченко М. М.				
Затв.		Жаріков Е. В.				



Зм.	Арк.	№ докум.	Підп.	Дата				
Розроб.		Клепіков С. Є.			Схема бази даних	Лит.	Арк.	Аркушів
Перев.		Храмченко М. С.						1
Т. Кон.						Аркуш		Аркушів
					Веб-сервіс для генерації та розгортання лендингів	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-93		
Н. Кон.		Головченко М. М.						
Затв.		Жаріков Е. В.						

КПІ.ІТ-9314.045440.07.99.СБД

КПІ.ІТ-9314.045440.07.99.СБД



КПІ.ІТ-9314.045440.08.99.СС

Зм.	Арк.	№ документа	Підпис	Дата	КПІ.ІТ-9314.045440.08.99.СС			
Розробив		Кленіков С. Е.			Схема структура класів програмного забезпечення	Літера	Маса	Масштаб
Перевірив		Храмченко М. С.						
Т. контр.						Аркуш	Аркушів	
Н. контр.		Голованенко М. М.			Веб-сервіс для генерації та розгортання пендінгів	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-93		
Затвердив		Жаріков Е. В.						