НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2024 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-застосунок для створення та редагування зображень

Виконав : студент __4__ курсу, групи _ІП-04_
(шифр групи)

_____Чоломбитько Кирило Олександрович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____ст. викладач Алещенко О. В._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) доцент, к.т.н. Волокита А. М._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _ст. викладач кафедри СПіСКС ФПМ_Радченко К. О._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2024 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Чоломбитько Кирила Олександровича

1. Тема проєкту *Веб-застосунок для створення та редагування зображень* керівник проєкту старший викладач Алещенко О. В..,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 27 травня 2024 року №2112-с

2. Термін здачі студентом закінченого проєкту *5 червня 2024 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Огляд існуючих рішень

Розділ 2. Огляд технологій та підходів в розробці системи.

Розділ 3. Огляд процесу розробки веб-застосунку.

Розділ 4. Огляд та аналіз розробленого проекту.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) компоненти системи(структурна схема), діаграма класів (функціональна схема), алгоритм дій системи (принципова схема).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Волокита А.М.		

7. Дата видачі завдання «05» січня 2024 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*20.02.2024-14.03.2024*	
2.	*Вивчення та аналіз завдання*	*14.04.2024-18.03.2024*	
3.	*Розробка архітектури та загальної структури системи*	*19.03.2024-24.03.2024*	
4.	*Програмна реалізація системи*	*01.04.2024-28.04.2024*	
5.	*Оформлення пояснювальної записки*	*29.04.2024-02.06.2024*	
6.	*Захист програмного продукту*	*03.06.2024*	
7.	*Передзахист*	*05.06.2024*	
8.	*Захист*	*19.06.2024*	

Студент-дипломник _____ Кирило ЧОЛОМБИТЬКО
(підпис)

Керівник проєкту _____ Олексій АЛЕЩЕНКО
(підпис)

АНОТАЦІЯ

У даній роботі був проведений огляд та аналіз існуючих веб-застосунків для створення та редагування зображень. Було проаналізовано їх переваги та недоліки. Спираючись на проведений аналіз та порівняльну таблицю, були виділені критерії які повинен задовольняти нове програмне забезпечення.

Була проведена робота над проектуванням бази даних, розробкою серверної та клієнтської частини системи. Новий веб-застосунок був створений використовуючи мову програмування – TypeScript. На клієнтській частині було використано React та Vite, на серверній NestJS.

Ключові слова: веб-застосунок для створення та редагування зображень, TypeScript, React, Vite, NestJS, docker-compose, PostgreSQL, TypeORM.

ANNOTATION

In this work, an overview and analysis of existing web applications for creating and editing images were conducted. Their advantages and disadvantages were analyzed. Based on the conducted analysis and comparison table, criteria that the new software should meet were identified.

During work process the design of the database, development of the server, and client parts of the system were implemented. The new web application was created using the programming language TypeScript. React and Vite were used on the client side, and NestJS was used on the server side.

Keywords: web application for creating and editing images, TypeScript, React, Vite, NestJS, docker-compose, PostgreSQL, TypeORM.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467100.002 ТЗ***	Веб-застосунок для	3		
			створення та редагування			
			зображень			
			Технічне завдання			
	А4	***ІАЛЦ.467100.003 ПЗ***	Веб-застосунок для	63		
			створення та редагування			
			зображень			
			Пояснювальна записка			
	А4	***ІАЛЦ.467100.004 Д1***	Компоненти системи	1		
			(структурна схема)			
	А4	***ІАЛЦ.467100.005 Д2***	Діаграма класів	1		
			(функціональна схема)			
	А4	***ІАЛЦ.467100.006 Д3***	Алгоритм дій системи	1		
			(принципова схема)			
	А4	***ІАЛЦ.467100.007 Д4***	Текст програмного коду	25		

					ІАЛЦ.467100.001 ОА		
Зм	***Лист***	***№ докум.***	***Підп***	***Дата***			
Розроб		Чоломбитько К. О.			*Веб-застосунок для створення та редагування зображень* Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перев		Алещенко О. В.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІП-04	

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Веб-застосунок для створення та редагування зображень*»

Київ – 2024

ЗМІСТ

					ІАЛЦ.467100.002 ТЗ		
		№ докум.	Підпис	Дата			
Розробив		Чоломбитько К.О.			**Веб-застосунок для створення та редагування зображень** Технічне завдання	Літ.	Аркуш 1 / Аркушів 3
Перевірив		Алещенко О. В.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІП-04	
Н. Контр.		Волокита А.М.					
Затвердив							

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку веб-застосунку для створення та редагування зображень, а також на подальшу підтримку та вдосконалення розробленої системи.

Областю застосування цієї системи є графічний дизайн, реклама та маркетинг, освіта.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка веб-застосунку для створення та редагування зображень, що забезпечить користувачів зручним, мінімалістичним та інтуїтивним інтерфейсом для вираження своїх творчих здібностей.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Простий і інтуїтивно-зрозумілий інтерфейс системи.
- Надати можливість користувачам робити маніпуляції з зображеннями.
- Надати можливість користувачам видаляти, додавати та змінювати проекти, що містять зображення. .

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- Встановлений веб-браузер.

5.3. Вимоги до апаратної частини

- RAM не менше ніж 8 ГБ.
- ROM не менше ніж 8 ГБ.
- Підключення до мережі Інтернет.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	20.02.2024 – 13.03.2024
Вивчення та аналіз завдання	14.03.2024 – 18.03.2024
Розробка архітектури та загальної структури системи	19.03.2024 – 24.03.2024
Програмна реалізація системи	01.04.2024 – 28.04.2024
Перевірка системи на працездатність та виправлення помилок	28.06.2024
Оформлення пояснювальної записки	29.05.2024 – 02.06.2024

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Веб-застосунок для створення та редагування зображень*»

Київ – 2024

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

UI	(User Interface) Користувацький інтерфейс
API	(Application Programming Interface) прикладни́й програмний інтерфейс
HTML	(HyperText Markup Language) Гіпертекстова мова розмітки
CSS	(Cascading Style Sheet) Каскадні таблиці стилів

ВСТУП

На сьогодні є величезна потреба у веб-застосунках для створення та редагування зображень. У давні часи люди користувалися підручними засобами для створення зображень, і з часом матеріали та інструменти для нанесення малюнків змінювалися та вдосконалювалися. Проте людство завжди стикалося з проблемою виправлення помилок у малюнках. Для виправлення зображень, зроблених олівцем, було винайдено гумку для стирання, а згодом з'явився коректор для виправлення ручки. Однак досі існують інструменти, такі як аерограф та пензлі, за допомогою яких майже неможливо прибрати помилку або зробити точні зміни. Я вважаю, що нанесення та виправлення малюнків підручними засобами в сучасному світі не є ефективним методом, бо, окрім того, що це займає невизначений час який витрачається не на користь, не можна бути впевненим в отриманні бажаного результату. Цифрові методи та інструменти для редагування економлять час та ресурси, дозволяючи користувачам прибирати помилки під час редагування та відстежувати зміни, робити більш точні маніпуляції з зображенням та мають більший наявний функціонал.

Ці інструменти використовуються майже в усіх сферах діяльності, а саме: в бізнесі, в рекламі, у сфері графічного дизайну, у сфері освіти, у фотографії, у видавництві, в кіно, у військовій сфері та інші. В нинішніх реаліях України є потреба такого застосунку в військовому напрямку, для створення військового контенту, який буде закликати іноземні спільноти допомагати нам. Він надасть можливість військовим редагувати карти: планувати маршрути, розміщувати стратегічно важливі об'єкти, візуалізувати можливі тактичні дії.

Веб-застосунок має на меті дати змогу користувачам редагувати та створювати зображення без завантаження програмного забезпечення та без використання їх особистих носіїв даних. Для цього система повинна мати певні

характеристики та властивості, тому буде реалізовано зручний користувацький інтерфейс, набір необхідних інструментів, серверна частина яка забезпечить збереження та менеджмент зображень, а також менеджмент користувачів, забезпечуючи високий рівень безпеки збережених даних використовуючи сучасні методи шифрування та аутентифікації.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Визначення понять

Розглядаючи основну концепцію веб-застосунку та його реалізацію, зокрема, у сфері створення та редагування зображень, надзвичайно важливо пояснити загальну архітектуру та специфічні функції, які характеризують ці платформи.

Застосунок — це комп'ютерна програма, яка написана людиною з метою розв’язання прикладних задач користувача. Це поняття було створене заради того, щоб було легше відрізняти програми, які виконують прикладні задачі від програм, які написані для функціонування операційної системи, системних утиліт та засобів і середовищ розробки тощо.[1]

Веб-застосунок, насамперед, відрізняється від звичайного застосунку. Він працює в основному використовуючи архітектуру клієнт-сервер.[2] Клієнт(не конкретна людина, а частина системи) представляє візуальну частину, доступ до якої зазвичай здійснюється через веб-браузер, або схожі програми, а сервер представляє серверну частину, яка обробляє запити клієнта, керує зберіганням і пошуком даних, працює з зовнішніми API тощо. Звичайний застосунок треба завантажувати та обновляти згідно вимог вашого персонального комп’ютера: оперативної системи, об’єму відеопам’яті, обсягу пам’яті тощо. На відміну від звичайного застосунку, веб-застосунки не треба завантажувати, вони є більш високодоступними та масштабованими, що є основними характеристиками в сучасному цифровому світі. Користувач взаємодіє з додатком через веб-інтерфейс, а обмін усіма необхідними даними здійснюється через глобальну мережу Інтернет, що забезпечує віддалений доступ і співпрацю в реальному часі, що є критично важливим у сучасних робочих процесах.

Зосереджуючись на веб-застосунках, призначених для створення та редагування зображень, ці платформи мають в наявності спеціалізовані інструменти, які відповідають потребам цифрових художників, дизайнерів, фотографів тощо. На відміну від звичайних веб-застосунків, програми, призначені для обробки зображень, включають розширені графічні можливості та алгоритми обробки зображень, які підтримують широкий спектр функцій від базових завдань редагування, таких як обрізка та зміна розміру, до більш складних операцій, таких як шарування, створення маски та цифрове компонування.

Інтерфейс користувача — це засіб мета якого забезпечити зручну взаємодію між користувачем (людиною) та системою. Засобів для реалізації користувацького інтерфейсу є дуже багато, наприклад: розташування елементів, зміна розміру, видимості, забарвлення, зміна розміру тексту, шрифту, різне положення вікон, розташування кнопок, форми вводу, розташування виводу інформації на екрані, меню тощо.[3]

Дизайн інтерфейсу користувача (UI) у програмах для створення та редагування відіграє вирішальну роль. Він має бути інтуїтивно зрозумілим і потужним, але не сильно складним дозволяючи користувачам різного рівня кваліфікації виконувати комплексні та складні завдання. Інтерфейс користувача зазвичай містить набір панелей інструментів та панелей які імітують середовище робочого столу, доступ до яких можна отримати через стандартний веб-браузер. Ця модель дизайну полегшує користування та час для того щоб навчитися використовувати застосунок, що корисно в освітніх і професійних умовах.

Безпека є ще одним важливим аспектом, особливо тому, що ці програми часто мають справу з конфіденційними правами інтелектуальної власності. Реалізація надійних заходів безпеки для захисту даних користувачів і запобігання несанкціонованому доступу є важливою. Ці заходи включають

захищені протоколи передачі даних, зашифровані рішення для зберігання даних і регулярні перевірки безпеки.

Підсумовуючи, веб-застосунки для створення та редагування зображень являють собою передовий перетин веб-технологій і графічного дизайну. Вони використовують фундаментальні принципи архітектури веб-застосунків, щоб забезпечити спеціалізовані функції, які відповідають потребам творчих професіоналів. Використовуючи переваги сучасних веб-технологій, ці програми забезпечують потужні, доступні та безпечні платформи для обробки цифрових зображень, демонструючи значну еволюцію способу взаємодії з цифровими медіа.

1.2 Огляд існуючих веб-застосунків для створення та редагування зображень

1.2.1 Canva

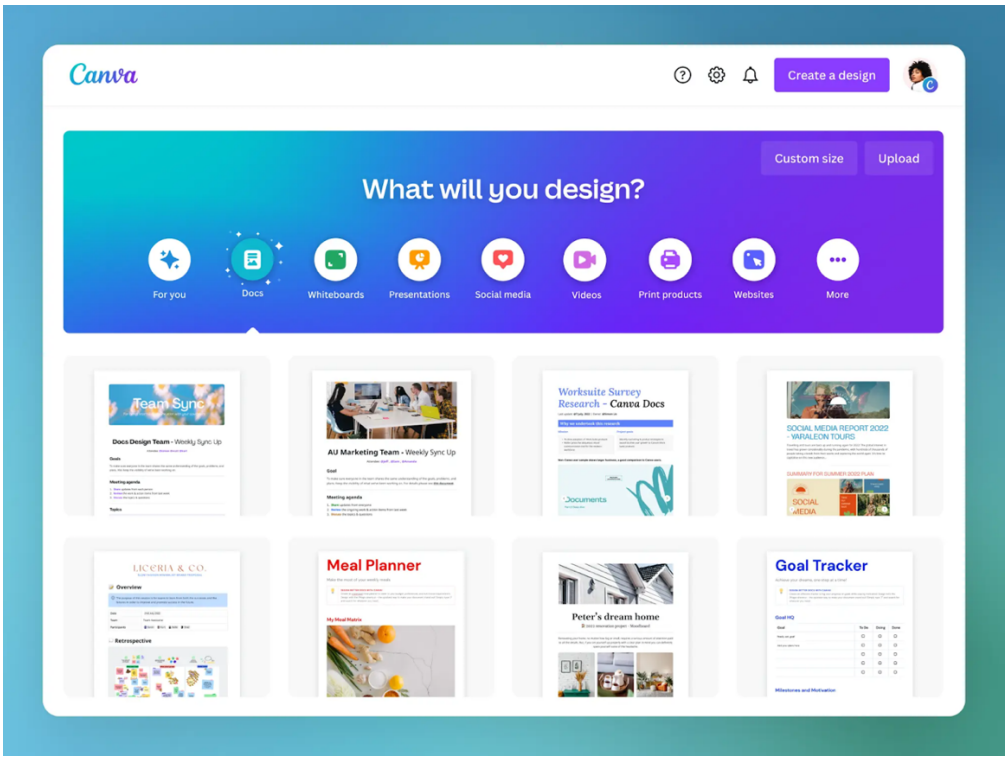


Рисунок 1.1 – Користувацький інтерфейс веб-застосунку Canva

Canva — це платформа спрямована на графічних дизайнерів або користувачів, які бажають створювати графіку, презентації, банери, афіші та інший візуальний контент для соціальних мереж. Доступна як веб-застосунок, так і мобільний застосунок. Сервіс пропонує велику базу існуючих шаблонів, зображень та шрифтів.[4]

Переваги цього веб-застосунку:

- Мультипристрійний доступ: має версії застосунку як у вебі, так і на мобільних пристроях. Це надає можливість користувачам майже в будь-який час та в будь-якому місці користуватись ним;
- Має в наявності багато тематизованих шаблонів та елементів дизайну, які вирішують багато загальновідомих прикладних графічних задач;

Недоліки цього веб-застосунку:

- Має в наявності багато преміум-функцій, за які треба платити підписку, через що користувачі які не можуть собі дозволити підписку, не матимуть можливості користуватися всіма перевагами програмного продукту.
- Складний користувацький інтерфейс: легкий в навігації, але не інтуїтивний та спроектований на більш освідчену аудиторію та творців. Всі необхідні інструменти знаходяться на видному місці, але їх дуже багато, що призводить до використання зайвих ресурсів та зусиль для освоєння цього застосунку.

1.2.2 Excalidraw

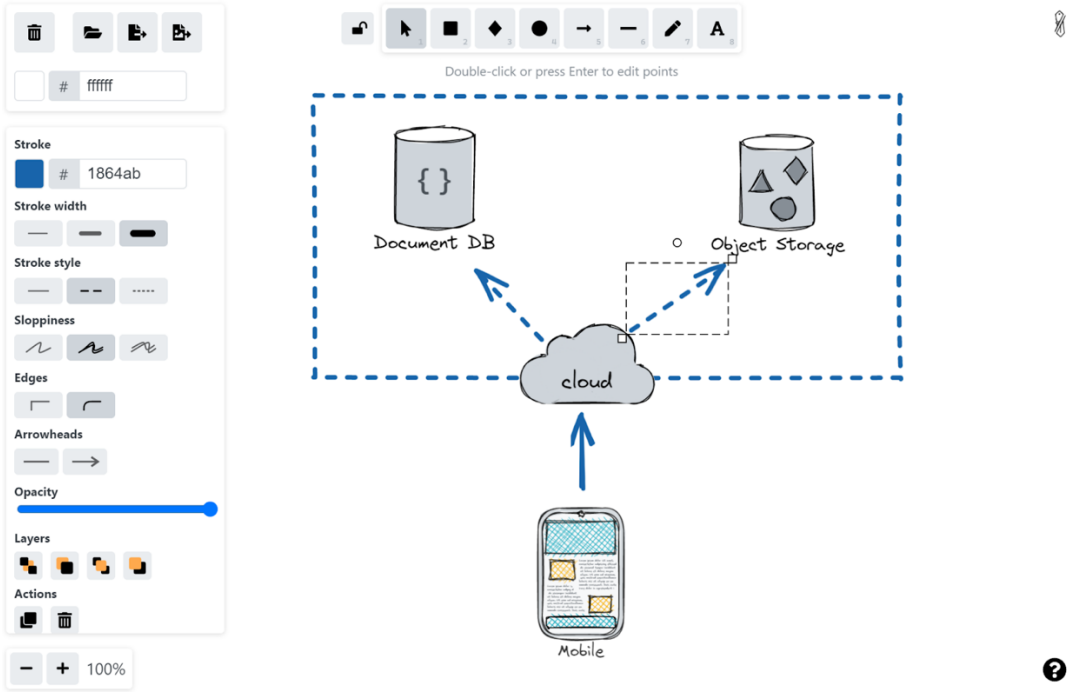


Рисунок 1.2 – Користувацький інтерфейс веб-застосунку Excalidraw

Excalidraw — це графічний редактор у вигляді віртуальної дошки, який спеціалізується на малюванні діаграм, намальованих від руки. Він пропонує нескінченну інформаційну панель на основі полотна з намальованим від руки стилем і підтримує різні функції.[5]

Переваги цього веб-застосунку:

- Дотримується політики відкритого коду: як програма з відкритим кодом, вона безкоштовна для використання, що може сподобатися широкій аудиторії, починаючи від художників-аматорів і закінчуючи професійними дизайнерами, які шукають економічно ефективне рішення. Користувачі не тільки можуть використовувати програмне забезпечення, але й мають можливість зробити свій внесок у його розвиток;
- Мінімалістичний, інтуїтивний користувацький інтерфейс: простий дизайн допомагає новим користувачам скоротити час навчання, полегшуючи їм початок використання програми без зайвих витрат ресурсів та часу. Це має вирішальне значення для підтримки широкої бази

користувачів, оскільки це вміщує користувачів із різними рівнями технічних навичок. Чистий сучасний інтерфейс може бути привабливим для користувачів, що може підвищити задоволеність користувачів і потенційно збільшити тривалість сеансів використання;

- Дозволяє зберігати користувацькі малюнки на їх серверах, а також шифрує їх, гарантує безпеку даних і приватність користувачів. Впровадження шифрування збережених даних гарантує, що малюнки та особиста інформація користувачів захищені від несанкціонованого доступу. Це величезна перевага в епоху, коли витоки даних є звичайним явищем, допомагаючи зміцнити довіру з базою користувачів.

Недоліки цього веб-застосунку:

- Обмежений та вузьконаправлений функціонал, який спрямований саме на малювання, створення діаграм, не надаючи можливість більш продвинутого редагування або застосування складних фільтрів, що може обмежити креативність та творчі здібності користувачів.

1.2.3 Figma

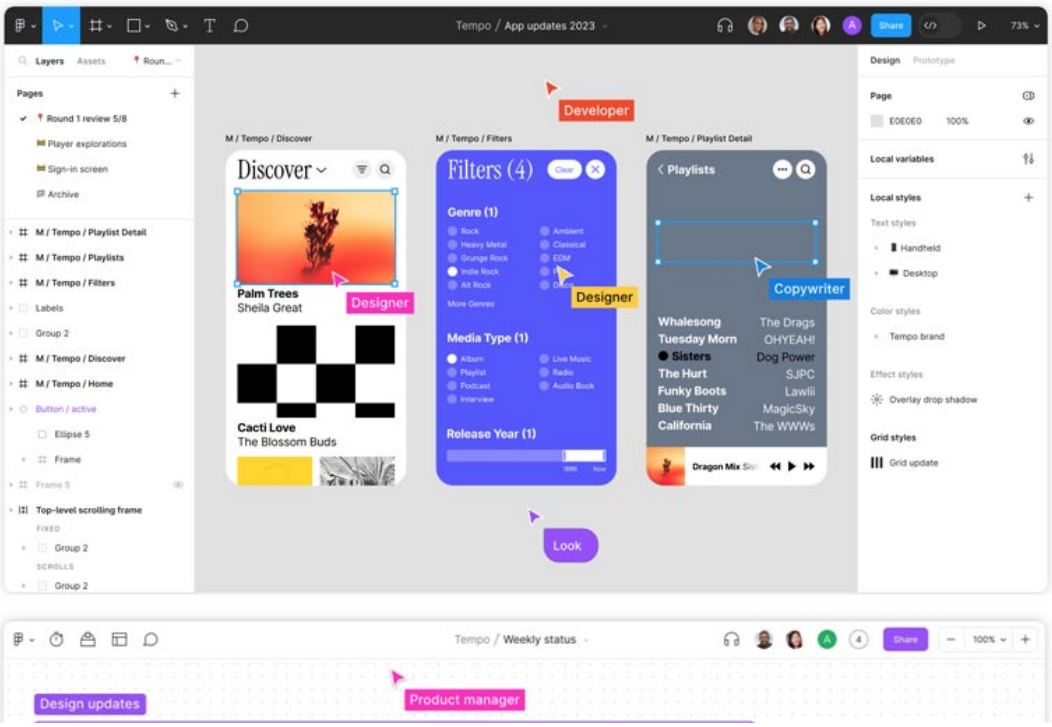


Рисунок 1.3 – Користувацький інтерфейс веб-застосунку Figma

Figma — це онлайн-сервіс розробки користувацьких інтерфейсів та прототипування. Використовує векторний тип графіки, тобто геометричних примітивів, що дозволяє зберігати якість зображення у різних розмірах. Має дві наявні версії: браузерна версія та як застосунок.[6]

Переваги цього веб-застосунку:

- Зручний та інтуїтивний користувацький інтерфейс: користувацький інтерфейс, спроектований згідно всіх стандартів UI/UX дизайну, спрямований на найменшу витрату зусиль та часу користуючись ним. Всі наявні інструменти та налаштування видно та зрозуміло те вони розташовані, допомогаючи користувачам легко розібратись в програмному застосунку;
- Наявні інструменти: надає широкий спектр загальновідомих та необхідних для роботи інструментів, завдяки яким є можливість будувати масштабовані та продвинуті UI/UX дизайни. Ці інструменти

задовольняють потреби UI/UX дизайнерів, дозволяючи їм реалізовувати майже повноцінно свої творчі ідеї.

Недоліки цього веб-застосунку:

- Дуже складний для художників-аматорів та нових людей у творчій сфері, через що треба буде витрачати багато часу та ресурсів для навчання та освоєння цього застосунку.
- Функціонал спрямований саме на UI/UX дизайнерів, а не загальну кількість творчих користувачів. Це може обмежувати привабливість для нових користувачів, або користувачів які мають бажання використовувати цей застосунок не тільки для дизайну користувацького інтерфейс;

1.3 Порівняння

Реалізуємо порівняльну таблицю на основі проведеного аналізу.

Таблиця 1.1 – Порівняльна таблиця існуючих веб-застосунків для створення та редагування зображень

Критерій	Canva	Excalidraw	Figma
Зручність UI	+	+	+
Простота використання	-	+	-
Наявні заходи безпеки	+	-	+
Відкритий код	-	+	-

Кінець таблиці 1.1

Критерій	Canva	Excalidraw	Figma
Повністю безкоштовний функціонал	-	+	-
Наявність української мови	-	+	-
Наявні мобільна версія	+	+	+

ВИСНОВОК ДО РОЗДІЛУ 1

Основуючись на порівняльну таблицю та аналіз аналогів, можна виділити декілька основних недоліків: усі аналоги занадто вузько направленні та спрямовуються на певну тематичну аудиторію, в більшості аналогів складний для нових користувачів інтерфейс, більшість не дотримуються політики відкритого коду, мають платний та складний функціонал, через що вони є недоступними та складними для аматорів. Водночас з усіх можна виділити переваги, а саме: зручний та інтуїтивний користувацький інтерфейс, що забезпечує швидкий та надійний доступ до наявного функціоналу застосунку, кожен з аналогів має мобільну версію, дозволяючи користувачам в різних ситуаціях користуватись їхнім застосунком та кожен з них має необхідні засоби безпеки, для збереження конфіденційності даних та зображень.

Після проведеного аналізу, можна прийти до висновку, що у новому веб-застосунку повинні бути: зручний користувацький інтерфейс, який може адаптуватися під різні види пристроїв(персональний комп’ютер, мобільний пристрій, тощо), більш загальний та простий функціонал, наявні заходи конфіденційності даних та українська локалізація. Також важливо зазначити, що застосунок необхідно зробити безкоштовним для збільшення кількості користувачів та конкурентної переваги на ринку.

РОЗДІЛ 2
ОГЛЯД ТЕХНОЛОГІЙ ТА ПІДХОДІВ В РОЗРОБЦІ СИСТЕМИ

2.1 Вибір архітектури системи

Основуючись на сучасні підходи та тенденції для розробки веб-застосунків було вирішено використовувати клієнт-серверну модель архітектури, оскільки вона є однією з найбільш поширених і добре зарекомендованих в сучасній розробці. Цей підхід забезпечує високий рівень гнучкості, масштабованості та зручності в управлінні, що є критичним для успішної реалізації сучасних веб-застосунків.

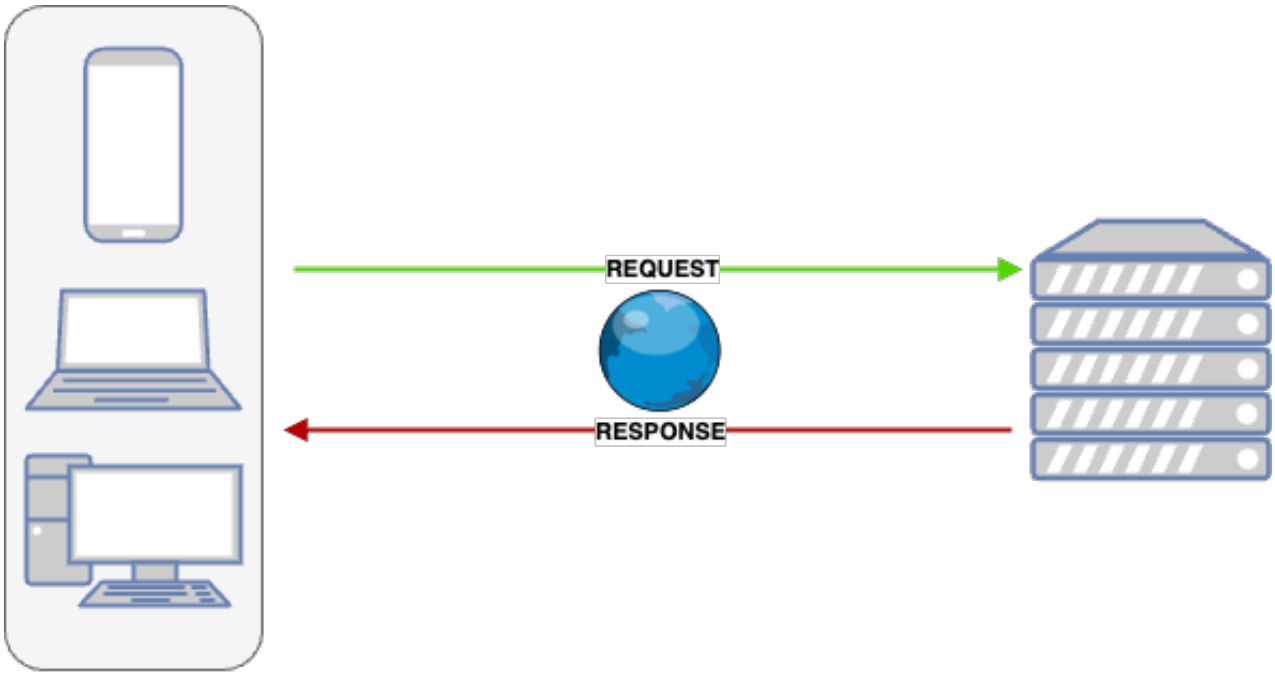


Рисунок 2.1 – Модель клієнт-серверної архітектури

Цей вид архітектури є одним з найрозповсюдженіших та надійних. Так як дозволяють централізовано керувати безпекою даних та розподілювати навантаження на сервер. Клієнт-серверна архітектура логічно від назви складається з двох головних компонентів системи, таких як клієнт та сервер.

Давайте розглянемо кожен компонент окремо:

- Клієнт - це комп’ютерна програма(зазвичай веб-браузер) або пристрій, завдяки якому користувач отримує доступ до необхідної йому інформації та ресурсів виконуючи запити на інший компонент системи - сервер. Зазвичай на клієнті користувач має зручний користувацький інтерфейс та весь необхідний функціонал для взаємодією з системою.
- Сервер - це пристрій або комп’ютерна програма або система з комп’ютерних програм. Сервер забезпечує обробку запитів від клієнта та надсилання відповідей, роботу з ресурсами та даними, захист та безпеку інформації, можливу взаємодію з іншими існуючими системами тощо.

Розглядаючи веб-застосунок для створення та редагування зображень, клієнт-серверна архітектура є найкращим вибором для розробки. Сервер забезпечить надійне збереження даних користувача, буде відповідати за аутентифікацію та шифрування даних, збереження зображень тощо. Клієнтська частина буде відповідати за інтуїтивну та правильну візуалізацію користувацького інтерфейсу, забезпечуючи користувача розумінням як користуватись застосунком та виконувати запити до сервера.

2.2 Вибір основних технологій для розробки

Перед початком розробки необхідно обрати всі технології та методи для вирішення задач. Хочу розпочати з вибору мови програмування, тому що це є основою для розробки, та є відправною точкою для вибору інших технологій та програмних рішень.

Найважливішими критеріями при виборі мови програмування є: тип застосунку, швидкість виконання, швидкість розробки, безпека, можливість розширення та суспільна підтримка.

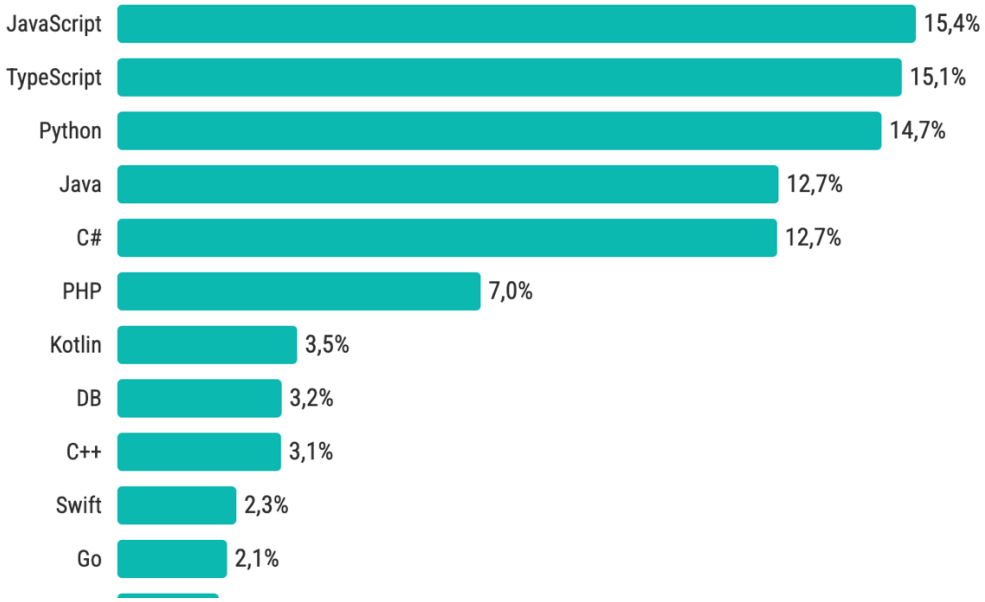


Рисунок 2.2 – Рейтинг мов програмування за версією українського ресурсу DOU.ua

Спираючись на український веб-ресурс DOU.ua найпопулярнішим рішенням при виборі мови програмування для реалізації застосунку є JavaScript. На мою думку, з вище зазначених мов найбільше для мого проекту підходить TypeScript спираючись на вимоги проекту та вищезазначені критерії.[7]

2.2.1 Мова програмування TypeScript

Рисунок 2.3 – TypeScript

TypeScript - статично типізована мова програмування, яка побудована на основі JavaScript та розробляється всім відомою компанією Microsoft.[8]

При виборі мови програмування для мого проекту я ретельно розглянув різні аспекти, які включають в себе не лише потреби проекту, але й специфіку

розробки та підтримки коду в майбутньому. Ось деякі ключові моменти, які спонукали мене обрати TypeScript, а не менш розширений JavaScript:

- По-перше, важливим фактором є статична типізація, яку надає нам TypeScript. Цей функціонал допомагає нам відловлювати помилки під час розробки програмного забезпечення, забезпечуючи розробника необхідною перевіркою на типи функцій, змінних та інших програмних сутностей. Це підвищує надійність коду та зменшує кількість багів і помилок.
- По-друге, підвищена читабельність коду. Типізація у цій мові програмування забезпечує розробника необхідною інформацією та анотаціями(стислою документацією) для програмних сутностей. Розробники використовуючи написані та типізовані частини коду, отримують додаткове розуміння поведінки програми, завдяки описаним очікуваним та вхідним типам даних. Це робить код більш легким для підтримки та розширення.
- По-третє, розвинена спільнота. Завдяки спільноті ми маємо у наявності дуже багато гарних існуючих рішень, а саме: бібліотек, фреймворків та кодових практик. Це покращує та пришвидшує процес розробки програмного забезпечення, завдяки використанню існуючих програмних рішень, а не написання з нуля.
- По-четверте, наслідує вже існуючі рішення, функції мови JavaScript. Цей пункт я вважаю є необхідним, через те що всі майже всі клієнтські застосунки використовують JavaScript для виконання коду.

Таким чином, мій вибір щодо TypeScript був обдуманим та обгрунтованим кроком для використання цієї мови як основної в проекті. Статична типізація, підвищена читабельність коду, розвинена спільнота та сумісність з JavaScript роблять TypeScript ідеальним вибором для досягнення мети проекту. Мова не лише відповідає нашим потребам, а й забезпечує надійність, ефективність та легкість у розробці та підтримці програмного

забезпечення. Я можу впевнено стверджувати, що TypeScript - це оптимальне рішення для втілення ідей та досягнення успіху у даному проекті.

2.2.2 Середовище запуску NodeJS

Рисунок 2.4 – NodeJS

Для того, щоб запустити TypeScript код, та розробляти застосунок, необхідно обрати надійне середовище запуску написаної програми, яка забезпечить ефективне виконня програми. Одним з найкращих та надійних рішень в даному питанні є NodeJS.

NodeJS - це середовище запуску мови програмування JavaScript з відкритим вихідним кодом. Він складається з двох частин: рушія JavaScript коду V8 та libuv.[9]. Рушій V8 відповідає за виконання JavaScript коду та автоматичне керування оперативною пам’ятю.

libuv - це бібліотека для запуску асинхронних операцій. Ця бібліотека надає дуже багато можливостей, саме у контексті NodeJS вона дозволяє виконувати не блокуючі операції, а саме: отримувати доступ та працювати з базою даних, обробляти запити на надсилати відповіді, працювати з файловою системою тощо. Він може ефективно обробляти кілька існуючих одночасних з’єднань. Завдяки виконанню не блокуючих операцій з’являється можливість використання NodeJS на сервері.

Використовуючи NodeJS можна створювати програмне забезпечення як для клієнту, так і для серверної частини. Це спрощує процес розробки повноцінного веб-застосунку, так як розробнику немає необхідності вивчати додаткові мови програмування та витрачати зайвий час та ресурси для розробки.

Node.js підтримує крос-платформний режим роботи. Це дозволяє програмістам розробляти та розгортати програму в різних операційних системах, включаючи найпопулярніші, таких як: Windows, macOS, Linux. Ця означає, що наші середовища розробки не є обмеженими операційною системою, що полегшує масштабування програми на різних платформах.

Отже, Node.js є ідеальним середовищем для розробки TypeScript застосунку, оскільки воно поєднує в собі високу продуктивність, ефективність роботи з асинхронними операціями та крос-платформність, що відповідає всім потребам у розробці та розгортанні програмного забезпечення.

2.2.3 NPM

Рисунок 2.5 – NPM

Після вибору мови програмування TypeScript та середовища запуску програмного коду NodeJS, постало питання встановлення фреймворків, бібліотек та необхідних залежностей для розробки. Для того щоб централізовано та організовано керувати усіма залежностями було вирішено використати NPM - NodeJS package manager.

NPM(NodeJS package manager) - найбільший у світі реєстр програмного забезпечення. Він складається з бази даних з програмним забезпеченням та

інформацією про нього, веб-застосунку з доступом до цієї бази та інтерфейсу командного рядка.[10]

Цей пакетний менеджер дає змогу відстежувати версії пакетів, оновлювати та завантажувати їх, запускати пакети та ділитися своїм програмним забезпеченням. Він пропонує можливість створення скриптів для автоматизації завдань, що робить розробку більш ефективною. Наприклад, можна налаштувати команди для автоматичного запуску тестів, збірки проекту, перевірки коду на відповідність стандартам тощо.

Завдяки великій кількості доступних пакетів та активній спільноті розробників, NPM є невід'ємною частиною сучасної розробки на платформі Node.js. Його використання дозволяє швидко адаптуватися до нових вимог та технологій, забезпечуючи гнучкість та ефективність у роботі.

Таким чином, NPM є необхідним для розробки нашого застосунку. Він надасть увесь необхідний функціонал для ефективного та організованого керування залежностями, що є ключовим фактором успішної реалізації проекту.

2.3 Вибір технологій для розробки клієнтської частини

Обираючи технології для клієнтської частини, потрібно звернути увагу на можливість технологій правильно відображати інформацію, робити запити до серверного програмного забезпечення та обробляти отриману інформацію, розташовувати елементи для формування користувацького інтерфейсу та змінювати їх вигляд для найкращого досвіду користування веб-застосунком.

2.3.1 HTML

Рисунок 2.6 – HTML 5

Побудування користувацького інтерфейсу є дуже важливою частиною веб-застосунку. Щоб забезпечити правильне розташування та наявність необхідних елементів інтерфейсу - треба обрати мову розмітки. HTML зарекомендував себе як надійний та універсальний інструмент.

HTML(HyperText Markup Language) - мова, яка використовується для задання структури веб-сторінки. Це мова розмітки, використовуючи яку можна додавати елементи на свою веб-сторінку.[11]

Ця мова є основою для відображення користувацького інтерфейсу в веб-браузерах. HTML складається з елементів, які використовуються для розташування та відображення їх на екрані, або для того, щоб задати їм певну поведінку.

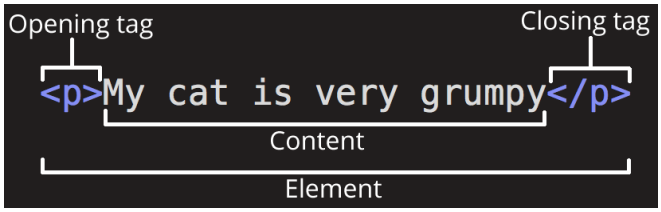


Рисунок 2.7 – Структура HTML-елемента

Ці елементи мають певну структуру:

- Відкриваючий/закриваючий тег - цей тег описує який саме елемент відобразити на веб-сторінці та де він починає діяти;
- Контент - це інформація яку отримує елемент для своєї роботи. На прикладі з малюнку це текст, але це може бути інший елемент, посилання тощо.

Основуючись на роботу веб-застосунків та веб-браузерів можна прийти до висновку, що мова розмітки HTML є необхідною для використання в розробці тому, що без неї не можна побудувати користувацький інтерфейс та відображати необхідну інформацію на клієнті.

2.2.4 CSS

Рисунок 2.8 – CSS

Ефективне розташування та стилізація елементів для формування користувацького інтерфейсу має велике значення. Для того, щоб забезпечити зручну та інтуїтивну взаємодію з веб-сайтом та покращити досвід користування – необхідно обрати мову стилізації. Єдиним фундаментальним та базовим рішенням є CSS.

CSS(Cascading Style Sheets) - це мова стилів для веб-контенту. Ця мова існує для стилізації та видозміни HTML-елементів.[12]

Технологія дозволяє видозмінювати та обирати які саме елементи отримають візуальні модифікації. Вибирання елементів для видозміни відбувається завдяки тегам, уніфікованим ідентифікаторам та класам(символьні ідентифікатори, які можна задавати елементам в HTML розмітці). Елементи можна групувати завдяки мові HTML.

Ця мова дозволяє робити такі операції над елементами, як:

- Видозміна: колір, фон, розмір, тінь, обводка тощо;
- Операції над текстом: зміна шрифту, товщини, розміру, кольору тощо;

- Розташування: зміна положення відносно осей X та Y, розташування в певних ділянках сторінки згідно світових стандартів(header, footer, main тощо);
- Межі та відступи між елементами та групами;
- Групові операції: розташування елементів відносно один одного(flex, grid тощо), групові видозміни тощо;
- Динамічні операції: анімації, переходи.

Отже, CSS є ідеальним рішенням проблеми стилізації в контексті розробки клієнтської частини, оскільки воно надає весь необхідний функціонал для реалізації зручного користувацького інтерфейсу та покращення досвіду користування.

2.2.5 Tailwind

Рисунок 2.9 – TailwindCSS

Мова CSS потребує дуже багато витраченого часу та зусиль від розробника. Для того, щоб покращити та пришвидшити досвід розробки та дати можливість програмістам виконувати більш важливі задачі - потрібно обрати CSS-фреймворк. Блискучим та зручним інструментом для цього є Tailwind.

Tailwind CSS - фреймворк, який покращує досвід розробки та стилізації. Він працює таким чином, що читає назви класів HTML-елементів та генерує статичний CSS-файл. Назви класів необхідно задавати згідно документації фреймворку.[13]

Головні переваги цього фреймворку:

- Надає багато існуючих рішень від спільноти програмістів та має велику аудиторію;

- Дуже швидко виконується завдяки попередньо згенерованих статичних CSS-файлів;

- Оптимізує код CSS завдяки чому зменшує кількість зайвих конструкцій, полегшуючи читання та розуміння;

- Легко інтегрується з сучасними технологіями розробки, такі як: React, Vue, Svelte тощо;

Отже, Tailwind є чудовим вибором CSS-фреймворку, оскільки він пришвидшує процес розробки, дозволяючи розробникам використовувати інтуїтивні назви класів та надає можливість витрачати час та ресурси на більш важливі задачі.

2.2.6 React

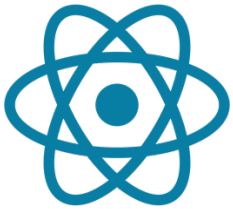


Рисунок 2.10 – React

HTML зазвичай складно структурувати та організовувати в логічні частини. Для того, щоб ефективно розробляти клієнтську частину та користувацький інтерфейс - нам необхідно використовувати компоненти. Однією з найкращих та зарекомендованих бібліотек для побудови та використання компонентів є React.

React - це бібліотека для ефективної побудови користувацьких інтерфейсів використовуючи компоненти. Компоненти - це окремі логічні частини(блоки) користувацького інтерфейсу. Це можуть бути як невеликі кнопки, так і повноцінні та масштабні сторінки. Компоненти дають можливість відображати інформацію динамічно, наприклад: отримати та відобразити інформацію з сервера, зовнішнього API тощо. Ще можна відображати різні компоненти в залежності стану веб-застосунку.[14]

В основі React лежить віртуальна об’єктна модель веб-сторінки. При завантаженні веб-сторінки у веб-браузер створюється об'єктна модель, яку веб-браузер використовує для коректного відображення інформації на екрані. Завдяки віртуальній об'єктній моделі React змінює тільки необхідні елементи на екрані, а не перерисовує всю веб-сторінку. Цей процес оптимізує роботу веб-застосунків на клієнті, що покращує досвід користування.

Таким чином, для кращого досвіду користування та досвіду розробки, React є гарним вибором бібліотеки. Він забезпечить нас необхідним API та засобами для побудови зручного та ефективного користувацького інтерфейсу.

2.2.7 Axios

Рисунок 2.11 – Axios

Для того, щоб спілкуватися з сервером, надсилати запити та обробляти отриману інформацію на клієнтській частині у проекті буде використовуватись Axios.

Axios - це бібліотека для роботи з HTTP протоколом побудована на промісах(асинхронна конструкція в JavaScript та TypeScript).[15] Axios надає весь необхідний функціонал, а саме:

- Створення запитів з клієнтської частини
- Додавати до HTTP запитів необхідні заголовки та інформацію
- Автоматично змінювати отриману інформацію під вимоги клієнтського застосунку(наприклад JSON)
- Дозволяє перехоплювати запити та відповіді для додаткових дій над ними(наприклад логування у терміналі розробника)

Отже, Axios є фундаментальною та необхідною бібліотекою у нашому застосунку. Завдяки ньому з’являється можливість спілкуватись з серверною частиною.

2.2.8 Konva

Рисунок 2.12 – Konva

Головною задачею та проблемою, є побудова функціоналу для створення та редагування зображень. Користувач повинен мати змоги додавати додаткові елементи на зображення та редагувати їх. Цю задачу вирішує фреймворк Konva.

Konva - це фреймворк, який розширює та покращує можливості існуючого браузерного HTML-елементу - canvas. [16] Він надає необхідний функціонал для створення сучасного веб-застосунку для маніпуляцій з зображеннями, а саме:

- Можливість керувати шарами(створювати, видаляти, змінювати порядок);
- Можливість відображати створені фігури;
- Можливість робити анімації, переходи тощо.

Елементи цього фреймворку можна використовувати з сучасними підходами для розробки користувацьких інтерфейс. Він буде гарно працювати з попередньо обраними технологіями.

Таким чином, вибір щодо Konva є необхідним для успішної розробки веб-застосунку. Він надасть необхідне API та функціонал для вирішення задач в контексті створення та редагування зображень.

2.2.9 Vite

Рисунок 2.13 – Vite

Тепер постала проблема, як скомпонувати усі технічні засоби, бібліотеки та встановлені залежності в один зручний програмний пакет, щоб усе запускалось та працювало правильно. Цю проблему вирішить Vite.

Vite - це інструмент побудови та компонування застосунків.[17] Він складається з двох основних частин:

- Сервер розробки, який перезавантажується набагато швидше ніж сервер React, що покращує досвід програміста;
- Побудовник, який компонує застосунок в один пакет;

Vite вирішує проблему компонування великих об’ємів модулів та пакетів. Ці модулі є фундаментальною частиною розробки на NodeJS і з ними можуть виникати проблеми несумісності та застарілих версій. Він автоматично керує залежностями та робить це більш правильно, ніж побудовник React. Vite працює з усіма сучасними фреймворками, що робить його конкурентним на ринку та корисним у використанні.

Таким чином, Vite є надійним та необхідним інструментом у розробці даного веб-застосунку. Він ефективно виконає усі поставлені перед ним задачі, а саме компонування та керування залежностями. А ще покращить досвід розробки завдяки своєму серверу.

2.4 Вибір технологій для розробки серверної частини

Обираючи технології для серверної частини, необхідно щоб вони забезпечили надійними та правильними обробками запитів, збереження та керування інформацією про користувачів, збереження та керування зображеннями та виконання задач аутентифікації.

2.4.1 NestJS

Рисунок 2.14– NestJS

Для того щоб ефективно працювати з базою даних, керувати користувачами та зображеннями, використовуючи у цей час всі необхідні та сучасні заходи безпеки(аутентифікація, шифрування) треба обрати серверний фреймворк. Найкращим рішенням на ринку на даний момент є NestJS.

NestJS - це фреймворк для ефективної побудови масштабованих та розширюваних серверних застосунків на основі технології NodeJS. Він повністю підтримує мову програмування TypeScript, що робить його підходящим для побудови нашого застосунку. NestJS надає дуже багато серверних абстракцій покращуючи цим досвід розробки та читаємість коду. Найголовнішу проблему, яку вирішує фреймворк - побудова правильної архітектури. Завдяки цьому серверні застосунки на основі NestJS будуть завжди масштабованими та легкими в підтримці.[18]

API цього фреймворку вирішує багато проблем, а саме:

- Аутентифікація: дозволяє використовувати сучасні методи та стратегії аутентифікування користувачів(наприклад JWT);

- Безпека та приватність даних: надає сучасні методи шифрування(хєшування) паролів завдяки модулю bcrypt;
- Робота з базою даних: надає можливість використовувати ORM для структурованого доступу до бази даних та дозволяє робити міграції.

Основуючись на можливості цього фреймворку, можна сказати що він вирішує всі необхідні задачі поставлені перед ним та гарно вписується для розробки веб-застосунку.

2.4.2 PostgreSQL

Рисунок 2.15 - PostgreSQL

Однією з головних задач серверної частини є ефективне зберігання та керування інформацією. Таку задачу в більшості випадків вирішує база даних. Для того, щоб правильно обрати базу даних, було зроблено невелике дослідження щодо сумісності з проектом та обраними технологіями. Внаслідок цього дослідження було обрано PostgreSQL.

PostgreSQL - це потужна, реляційна база даних, яка зарекомендовала себе за надійність, продуктивність та ефективність. Вона використовує мову запитів SQL, яку використовує більшість реляційних баз даних. Ця мова та база підтримують усі необхідні операції для того, щоб серверна частина працювала правильно. Має дуже велику базу розширень та модулів, для того щоб вирішувати різні задачі та проблеми. Ці модулі розвиваються великою спільнотою програмістів, тому що PostgreSQL дотримується політики відкритого ходу, що доводить її надійність. Підтримує дуже багато типів даних, таких як: числа, строки, булеві значення, дата, масив тощо.[19]

Щоб використати PostgreSQL з серверним фреймворком NestJS - було обрано бібліотеку “pg”. Ця бібліотека є необхідною для правильного функціонування та роботи бази з серверним застосунком.

Таким чином, вибір бази даних PostgreSQL є гарним та обгрунтованим, основуючись на поставлені задачі перед серверною частиною та проектом вцілому.

2.4.3 TypeORM

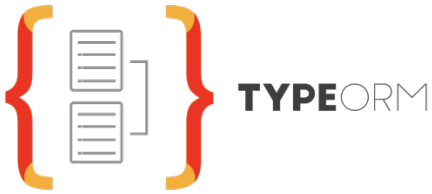


Рисунок 2.16 – TypeORM

Після вибраного фреймворку та бази даних, необхідно обрати технологію яка дозволить у серверному застосунку структуровано та організаційно працювати з базою даних. Це потрібно для того, щоб не було проблем з явними запитами. Такі проблеми вирішує технологія під назвою об’єктно-реляційне відображення. Було вирішено використати TypeORM.

TypeORM - це програмне забезпечення яке надає розробникам можливість об’єктно-реляційного відображення. Його можна використати в NodeJS та NestJS застосунку. Ця технологія підтримує дуже багато функцій, такі як міграції, створення сутностей, свя’зків, репозиторії, логування, підтримка бази даних PostgreSQL тощо.[20]

Таким чином, для правильного функціонування серверного застосунку та роботи з базою даних, TypeORM є найкращим рішенням.

2.4.4 Docker та docker-compose

Рисунок 2.17 – Docker

Під час розробки веб-застосунку для створення та редагування зображень є проблема яку треба вирішити. Проект використовує дуже багато необхідних технологій таких як серверна частина на NestJS, база даних PostgreSQL та клієнтська частина на основі React. Кожен компонент цього проекту має залежності, вимоги та версії, через що процес розробки стає більш комплексним та складним. Для вирішення вищепоставленної проблеми нам треба бути впевненими в роботі на різних платформах розробки та в різний час. Треба бути впевненими в тому що проект буде працювати як і під час процесу розробки так і під час процесу роботи, розгортання і масштабування. Таку проблему може вирішити Docker та docker-compose.

Docker - це програмне забезпечення для контейнеризації застосунків. Його принцип роботи побудований на тому, що він створює контейнер з застосунком, потім який можна запускати в різних середовищах. Цей контейнер завжди буде мати ті версії залежностей, які використовувались при розробці, забезпечуючи розробників без проблемним розгортанням та масштабування проекту. Застосунки запускаються використовуючи легку віртуальну машину у самому Docker, таким чином це є оптимізованим рішенням, тому що контейнер тільки містить інформацію про середовище запуску, а не саме середовище, через що вони б займали дуже великий обсяг пам’яті.[21]

docker-compose - це утиліта для керування та запуском водночас декілька контейнерів. Вона є необхідною для процессу розробки, щоб можна було запускати одразу всі необхідні компоненти(серверну частину, клієнтську частину та базу даних) та бачити роботу веб-застосунку загалом.

Таким чином, Docker та docker-compose покращать процес розробки та забезпечують стабільність середовища

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі було проаналізовано та обґрунтовано вибір технологій, архітектури, фреймворків, бази даних та мови програмування в контексті розробки веб-застосунку для створення та редагування зображень.

З урахуванням вимог проекту та сучасних стандартів розробки програмного забезпечення, було обрано клієнт-серверну модель як архітектурне рішення. Так, як це найкращий вибір для веб-застосунку. Вона добре підходить для масштабованості, ефективності та гнучкості, що є дуже важливими аспектами будь-якої сучасної програми.

Для реалізації проекту було приділено велику увагу вибору базових технологій як для клієнта, так і для сервера. Використання TypeScript, NodeJS як основу розробки забезпечить нас усіма необхідними можливостями. Щоб забезпечити оптимальну продуктивність та легкість розробки зручного користувацького інтерфейсу, використовується бібліотека React, фреймворк Tailwind CSS та інші сучасні технології. Вибір серверних технологій, таких як NestJS і PostgreSQL, у поєднанні з використанням TypeORM для взаємодії з базами даних забезпечує надійність, швидкість і легкість масштабування та обслуговування.

Інтеграція Docker і docker-compose у проект дозволяє створювати ізольовані контейнери для додатку та його залежностей. Це дозволяє забезпечити цілісність середовища розробки та розгортання, що допомагає уникнути проблем, пов'язаних із сумісністю та конфліктами версій програмного забезпечення. Крім того, Docker дозволяє легко масштабувати додаток та розгортати його у різних середовищах, що є важливим аспектом у сучасному розробці програмного забезпечення.

Таким чином, вибір архітектури та технологій було оптимізовано відповідно до потреб проекту з точки зору гнучкості, ефективності, швидкості впровадження та простоти розробки при забезпеченні високого рівня безпеки та стабільності роботи веб-застосунку.

РОЗДІЛ 3
ОГЛЯД ПРОЦЕСУ РОЗРОБКИ ВЕБ-ЗАСТОСУНКУ

3.1 Початкове налаштування компонентів та технологій системи

3.1.1 Серверна частина

Перед початком розробки проекту нам необхідно встановити обране середовище запуску, а саме NodeJS. Для цього нам треба зайти на офіційний сайт та завантажити виконуваний файл, який відкриває нам установщик. Або, як було у моєму випадку, використати менеджер пакетів вашої операційної системи.

```
kcholombytko@mbp1 ~ % brew install node
```

Рисунок 3.1 - Команда, яка завантажує NodeJS, використовуючи пакетний менеджер Brew, на операційній системі MacOS.

Для перевірки на успішність завантаження середовища запуску, нам необхідно виконати команду “node -v”. В результаті успішного встановлення, результатом цієї команди буде версія встановленого середовища запуску.

```
kcholombytko@mbp1 ~ % node -v
v20.8.0
```

Рисунок 3.2 - Результат успішного виконання команди “node -v”

Наступним логічним кроком буде встановлення всіх необхідних фреймворків та залежностей проекту. Перед цим нам треба встановити NPM - Node package manager. При встановленні середовище запуску NodeJS, npm автоматично завантажується та встановлюється, ми можемо це перевірити командою “npm -v”.

Рисунок 3.3 - Результат успішного виконання команди “npm -v”

Тепер встановимо необхідні нам залежності та фреймворки, а саме: NestJS, pg, typeorm. Почнемо з встановлення NestJS, для цього нам треба виконати команду “npm i -g @nestjs/cli”. Результатом буде встановлений інтерфейс командного рядка NestJS, завдяки якому ми можемо створити проект на базі обраного фреймворку. Створимо проект, виконавши команду “nest new <project_name>.

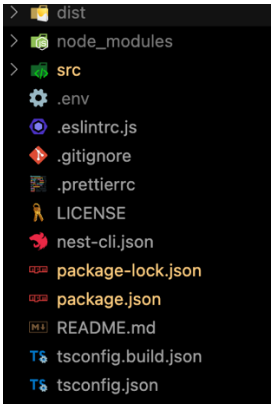


Рисунок 3.4 - Результат виконання команди “nest new <project_name>”

В результаті ми можемо побачити згенеровані фреймворком директорії та файли:

- dist: містить файли, які генеруються та запускаються в результаті побудування проекту;
- node_modules: містить усі файли залежностей проекту;
- src: містить файли у яких буде написаний код під час розробки проекту
- файли у головній директорії проекту: містять конфігураційну інформацію, для інструментів, які використовують серверні застосунки на базі NodeJS та NestJS.

Тепер встановимо залежності для роботи з базою даних використавши npm у директорії з проектом. Виконаємо команду “npm i typeorm pg @nestjs/typeorm @nestjs/config”. Завдяки цій команді ми встановили необхідні залежності для роботи з базою даних PostgreSQL, TypeORM.

У цій частині ми успішно зробили початкове налаштування серверного компоненту системи. Ми встановили всі необхідні залежності та створили початкову версію проекту.

3.1.2 Клієнтська частина

Розпочнемо початкове налаштування клієнтської частини. За-ініціалізуємо проект використавши Vite, npm та React. Для цього треба виконаємо команду в командному рядку.

```
kcholombytko@mbp1 diploma % npm create vite@latest frontend -- --template react

Scaffolding project in /Users/kcholombytko/Developer/diploma/frontend...

Done. Now run:

  cd frontend
  npm install
  npm run dev
```

Рисунок 3.5 - Результат успішного виконання команди для ініціа- лізації проекту Vite

Рисунок 3.6 - Структура проекту в результаті успішної ініціалізації

Вище можемо побачати директорії, які були створені:

- public: містить набір статичних файлів, які використовуються клієнтським застосунком;
- src: містить набір файлів(компонентів), які будуть створені під час розробки;
- файли у головній директорії проекту: набір конфігураційних файлів.

Наступним кроком є встановлення залежностей та бібліотек, які будуть використані під час розробки веб-застосунку. Спочатку треба встановити CSS-фреймворк Tailwind. Користуючись офіційною документацією цього фреймворку, нам спочатку треба завантажити його використавши npm. Для цього виконаємо команду “npm i -D tailwindcss”. Після цього треба запустити виконуваний файл цього фреймворку виконавши команду “npx tailwindcss init”. В результаті цієї команди буде успішно встановлено tailwind, а також буде створено конфігураційний файл цього фреймворку.

```
/** @type {import('tailwindcss').Config} */
export default {
  content: [   ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Рисунок 3.7 – Згенерований конфігураційний файл TailwindCSS.

Для того, щоб цей фреймворк правильно працював, треба додати в content шлях до компонентів.

```
content: [
  "./index.html",
  "./src/**/*.{js,jsx,ts,tsx}"
],
```

Рисунок 3.8 – Прописані шляхи до компонентних файлів.

В результаті цієї конфігурації, ми зробили все необхідне для того, щоб фреймворк працював правильно. Перейдемо до встановлення інших залежностей - Axios та Konva. Axios використовується для виконання HTTP-запитів з клієнтської частини, а Konva для роботи з браузерним полотном. Для встановлення цих залежностей, нам треба виконати команду “npm i axios konva react-konva”.

```
added 5 packages, and audited 374 packages in 966ms

127 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
kcholombytko@mbp1 frontend %
```

Рисунок 3.9 - Результат встановлення залежностей.

Таким чином, завдяки нашим діям було ініціалізовано клієнтський проект використовуючи Vite та React. Було завантажено необхідні фреймворки та бібліотеки для побудування користувацького інтерфейсу.

3.1.3 Docker та docker-compose

Перед початком розробки залишились тільки налаштувати docker та docker-compose. Для цього нам треба створити Dockerfile у директорії з серверним застосунком.

Створимо Dockerfile у директорії серверного компоненту та сконфігуруємо його.

```
FROM node:20
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
CMD [ "npm", "run", "start:dev" ]
```

Рисунок 3.10 - Конфігураційний файл Docker у серверній частині проекту

У цьому файлі описано:

- Використання NodeJS останної LTS-версії(20)
- Створення у контейнері робочії директорії
- Встановлення усіх залежностей у контейнері
- Запуск веб-застосунку

Після правильного налаштування Dockerfile, нам необхідно налаштувати docker-compose. Для цього створимо файл docker-compose.yml. Цей файл буде використовуватись самою утилітою та в ньому буде написана конфігурація для контейнерів та образів для системи Docker.

```yaml
version: '3.8'

services:
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    volumes:
      - ./backend:/app
      - /app/node_modules
    environment:
      - PORT=3000
      - HOST=db
      - DB_PORT=5432
      - USERNAME=test_user
      - PASSWORD=test_password123
      - DB_NAME=diploma
      - JWT_SECRET=test_jwt
      - SALT=$$2b$$10$$1dpR6pNBwFGtTYUw6cPr9u
    depends_on:
      - db

  db:
    image: postgres:14
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_DB=diploma
      - POSTGRES_USER=test_user
      - POSTGRES_PASSWORD=test_password123
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

Рисунок 3.11 – Конфігураційний файл docker-compose.yml

У цьому конфігураційному файлі описано як побудувати сервіс серверної частини на основі створеного вище Dockerfile та змінні середовища, для коректної роботи застосунку. А також було описано створення контейнеру з базою даних PostgreSQL та її конфігурація.

3.2 Розробка серверної частини

3.2.1 Опис сутностей у базі даних

Одним з найголовнішим компонентом веб-застосунку є база даних. Для проектування бази даних необхідно визначити в вигляді яких сутностей(таблиць) будемо зберігати наші дані. Основуючись на потребах в керуванні користувачами та зображеннями було обрані такі сутності:

- Користувач - сутність яка буде використовуватись для збереження інформації про користувача;
- Проект - сутність яка буде містити загальну інформацію про існуючі проекти. Вона буде містити звя’зок до користувача;
- Зображення - сутність яка буде містити інформацію про зображення. Воно буде містити зв’язок до проекту;

Таблиця users(користувачі) буде основною у нашій базі даних, вона міститиме інформацію про кожного користувача, який буде користуватись веб-застосунком. Ця інформація буде використовуватись для аутентифікації та функціонування застосунку.

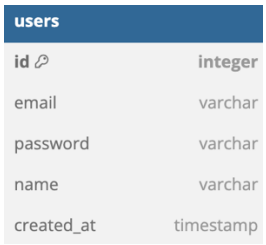


Рисунок 3.12 - Структура таблиці користувачів

Таблиця projects(проекти) буде містити інформацію про проекти які наявні у користувачів. Вона надасть змогу орієнтуватися між проектами, над якими працював користувач, та буде містити в собі зв’язок до зображення.

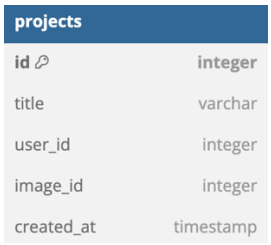


Рисунок 3.13 - Структура таблиці проектів

Таблиця images(зображення) буде містити саме зображення у base64 форматі, для того щоб можна було ефективно відправляти з серверу збережене зображення, та можна було конвертувати у різні формати для завантаження.

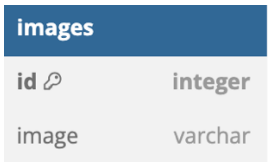


Рисунок 3.14 - Структура таблиці зображень

3.2.2 Розробка серверного застосунку

Під час розробки веб-застосунку було прийнято рішення щодо створення API з трьома логічними ендпоінтами: аутентифікацією, проектами та зображеннями.

Перед початком роботи, необхідно у головному модулі серверного застосунку налаштувати TypeORM. Цього можна досягти використавши метод модулю TypeOrmModule - forRootAsync. Як аргумент цей метод приймає об’єкт з налаштуваннями, такими як: порт, назва бази даних, шлях до написаних сутностей тощо. Було прописано використання параметрів з змінних середовища, використовуючи модуль та сервіс “@nestjs/config”.

```
TypeOrmModule.forRootAsync({
  imports: [ConfigModule],
  inject: [ConfigService],
  useFactory: (configService: ConfigService) => ({
    type: 'postgres',
    host: configService.get('HOST'),
    port: configService.get('DB_PORT'),
    username: configService.get('USERNAME'),
    password: configService.get('PASSWORD'),
    database: configService.get('DB_NAME'),
    entities: [User, Project, Image],
    synchronize: true,
  }),
```

Рисунок 3.15 - Використання методу forRootAsync у серверному застосунку

Після налаштування бази даних в серверному застосунку було розпочато роботу над реалізацією основного функціоналу, а саме аутентифікації.

Для роботи аутентифікації нам потрібен сервіс, який працюватиме з базою даних, а саме з таблицею користувачів. Перед створенням сервісу, було прописано сутність користувачів з бази даних для роботи TypeORM. Для цього було створено файл user.entity.ts. Важливо зазначити, що повинен бути наявний зв'язок до сутності проектів та у поля email необхідно поставити налаштування unique - true. Це налаштування перевіряє, щоб кожен користувач мав унікальну електронну поштову адресу.

```
@Entity('user')
export class User {
  @PrimaryGeneratedColumn()
  public id: number;

  @Column()
  public name: string;

  @Column({ unique: true })
  public email: string;

  @Column()
  public password: string;

  @OneToMany(() => Project, (project) => project.user)
  projects: Project[];
}
```

Рисунок 3.16 - Реалізація сутності користувач у проекті

Після створення файлу сутності, було реалізовано логіку сервісу користувачів у файлі user.service.ts. Необхідний функціонал для реалізації аутентифікації надають нам методи класу UserService: create, findOne.

Ці методи працюють з TypeORM та сутністю користувач, використовуючи методи фреймворку. Тепер при створенні сервісу аутентифікації ми зможемо перевірити чи існує користувач в системі та створити нового.

Аутентифікація була реалізована використовуючи JWT-токен. Під час процесу реєстрації в системі, ми перевіряємо чи існує вже такий користувач, якщо ні - створюємо нового користувача та відправляємо JWT-токен у відповідь і зберігаємо його на клієнтській частині. При логіні ми верифікуємо JWT-токен та перевіряємо чи існує користувач в системі, якщо так - повертаємо успішну відповідь. Для перевірки на JWT-токен під час запиту було використано спеціальний клас з фреймворку NestJS - guard. Головною функцією цього класу є перевірка запиту по критеріям, які нам необхідні, якщо дані з запиту пройшли перевірку - продовжити виконання запиту, якщо ні - відправити помилку.

```
@Injectable()
export class AuthGuard implements CanActivate {
  constructor(
    private jwtService: JwtService,
    private userService: UserService,
  ) {}
  async canActivate(context: ExecutionContext): Promise<boolean> {
    const request = context.switchToHttp().getRequest();
    const token = this.extractTokenFromHeader(request);

    if (!token) throw new AuthTokenNotFoundException();

    const payload = await this.jwtService.verifyAsync(token);

    if (!payload) throw new InvalidTokenException();

    const user = await this.userService.findOne({ email: payload.email });

    if (!user) throw new UserNotFoundException();

    return true;
  }

  private extractTokenFromHeader(request): string | undefined {
    const [type, token] = request.headers.authorization?.split(' ') ?? [];
    return type === 'Bearer' ? token : undefined;
  }
}
```

Рисунок 3.17 - Клас Guard, який перевіряє JWT-токен на правильність при надходженні нового запиту

У цьому класі ми беремо токен для аутентифікації з заголовків запиту, потім перевіряємо його на правильність та перевіряємо чи існує такий користувач у системі, якщо по нашим критеріям запит проходить - повертаємо істинне значення, після чого запит продовжується. Для його використання на HTTP-маршрутах було додано декоратор @UseGuard(AuthGuard). Завдяки цьому декоратору клас Guard виконує роботу.

Після реалізації аутентифікації було створено файл project.entity.ts. У цьому файлі була описана сутність проекту, а також зв’язки до сутності користувачів та сутності зображень.

```
@Entity('project')
export class Project {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ unique: true })
  title: string;

  @ManyToOne(() => User, (user) => user.projects)
  user: User;

  @OneToOne(() => Image, { cascade: true })
  @JoinColumn()
  image: Image;
}
```

Рисунок 3.18- Реалізація сутності проекту.

Після створення сутності, одразу було розпочато роботу над сервісом ProjectService. У цьому сервісі були реалізовані базові операції над проектами - отримати всі проекти для користувача, отримати інформацію про конкретний проект, змінювати та видаляти проект.

Для того, щоб можна було оперувати проектами було створено ProjectController. Використовуючи декоратор @Controller(“project”) ми створили маршрут до сервісу, який працює з проектами. Тепер є можливість надсилати HTTP-запити на “/project”.

```
@Controller('project')
export class ProjectController {
  constructor(private readonly projectService: ProjectService) {}

  @Post()
  create(@Body() createProjectDto: CreateProjectDto) {
    return this.projectService.create(createProjectDto);
  }

  @Get('/findByUserId/:id')
  findAll(@Param('id') id: string) {
    return this.projectService.findAllByUserId(+id);
  }

  @Get(':id')
  findOne(@Param('id') id: string) {
    return this.projectService.findOneById(+id);
  }

  @Patch(':id')
  update(@Param('id') id: string, @Body() updateProjectDto: UpdateProjectDto) {
    return this.projectService.update(+id, updateProjectDto);
  }

  @Delete(':id')
  remove(@Param('id') id: string) {
    return this.projectService.remove(+id);
  }
}
```

Рисунок 3.19 – ProjectController.

Таким самим чином, було створено сутність зображення. Описавши звя’зок до сутності проектів, ми правильно описали відношення між сутностями.

```
@Entity('image')
export class Image {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  image: string;

  @OneToOne(() => Project, (project) => project.image)
  project: Project;
}
```

Рисунок 3.20 - Сутність зображення.

По аналогії до сервісу проектів, було реалізовано ImageService та ImageController. ImageController виконує необхідну маршрутизацію HTTP-запитів, а ImageService надає нам весь функціонал для створення, редагування та видалення зображень у базі даних.

3.3 Розробка клієнтської частини

Важливим аспектом клієнтської частини системи є створення запитів до серверу та коректна обробка відповідей. Перед початком реалізації компонентів, нам треба реалізувати аутентифікацію в системі.

Для надсилання HTTP-запитів до серверу було використано бібліотеку Axios. Використовуючи метод бібліотеки Axios - post, ми створюємо HTTP POST запит по необхідному нам маршруту. Передавши в аргументи маршрут аутентифікації “<SERVER_URL>/auth/signUp” та тіло, яке задовольняє інтерфейс, надсилається запит та ми отримуємо відповідь.

```
interface ISignUp {
  name: string;
  email: string;
  password: string;
}
```

Рисунок 3.21 - Опис структури тіла запиту на маршрут реєстрації в системі.

За таким самим принципом, було створено відправник запиту для логіну на маршрут “<SERVER_URL>/auth/signIn”.

```
interface ISignIn {
  email: string;
  password: string;
}
```

Рисунок 3.22 - Опис структури тіла запиту на маршрут логіну в системі.

Для того, щоб використовувати наші отримані дані в усіх компонентах (user_id, username тощо) було використано функцію React - контекст. Контекст це механізм, який дає можливість передавати в інші компоненти інформацію. У нашому застосунку реалізовано контекст Auth, який передає інформацію до інших компонентів про аутентифікованого користувача, таку як: ідентифікатор,

ім’я користувача, електронна поштова адреса. Використовуючи функцію CreateContext ми створили контекст, у інших компонентах його можна використати завдяки функції useContext(Auth).

Перейдемо до компонентної структури.

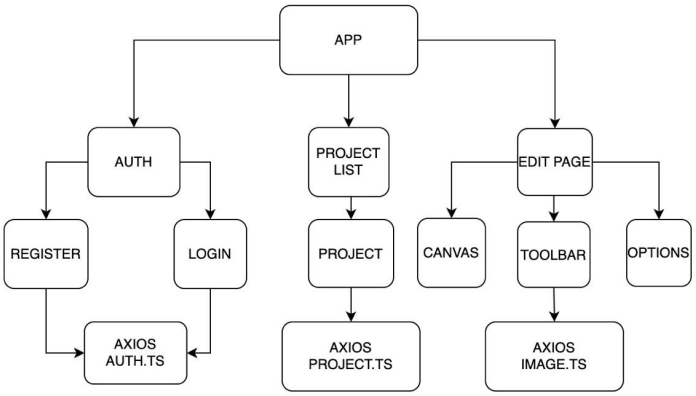


Рисунок 3.23 - Структура розроблених компонентів клієнтської частини застосунку.

Компоненти:

- Auth: цей компонент відповідає за аутентифікацію в системі. Використовує два компоненти: Register, Login. Також використовує написаний відправник запиту, який описаний у файлі auth.ts
- ProjectList: цей компонент відповідає за відображення списку проектів, використовує компонент - Project(показує один певний проект). Працює з методом для відправки запиту з файлу project.ts
- EditPage: Основний компонент нашого застосунку, цей компонент дає можливість змінювати зображення(створене або існуюче), зберігати на сервері, використовуючи методи з файлу image.ts

ВИСНОВОК ДО РОЗДІЛУ 3

У розділі 3 було описано повний процес розробки веб-застосунку для створення та редагування зображень. На початку ми налаштували усі необхідні інструменти перед початком роботи, а саме NodeJS та npm. Використовуючи ці інструменти та технології ми почали розробку серверної частини.

Протягом розробки серверної частини було визначено та описано сутності системи. Основуючись на ці сутності ми реалізували таблиці в базах даних для структурованого збереження необхідної інформації. Розробляючи серверну частину на основі NestJS та TypeORM ми правильно описали роботу з базою даних, створили аутентифікацію та обмеження доступу до них, для того щоб не було проблем з конфіденційністю. Також було розроблено доступ до сервісів які працюють з даними та інформацією, для забезпечення взаємодії з клієнтською частиною. Було налаштовано docker та docker-compose для правильної та ізольованої роботи серверної частини, та усунення проблем під час розробки.

Після створення серверної частини, було проведено роботу над реалізацією клієнтської частини системи. Було налаштовано Vite для усунення можливих проблем з залежностями системи та забезпеченням можливостями бібліотеки React під час розробки. Також було налаштовано фреймворк TailwindCSS, що спростив роботу над стилізацією клієнтського застосунку. Після усіх необхідних налаштувань, були реалізовані функції для взаємодії з сервером завдяки Axios. Потім було побудовано та реалізовано компонентну структуру.

РОЗДІЛ 4
ОГЛЯД ТА АНАЛІЗ РОЗРОБЛЕНОГО ПРОЕКТУ

У цьому розділі буде описано процес користування веб-застосунком та перевірка правильності його роботи.

Уявімо, що новий користувач зайшов в систему. Перше, що він бачить це вікно аутентифікації.

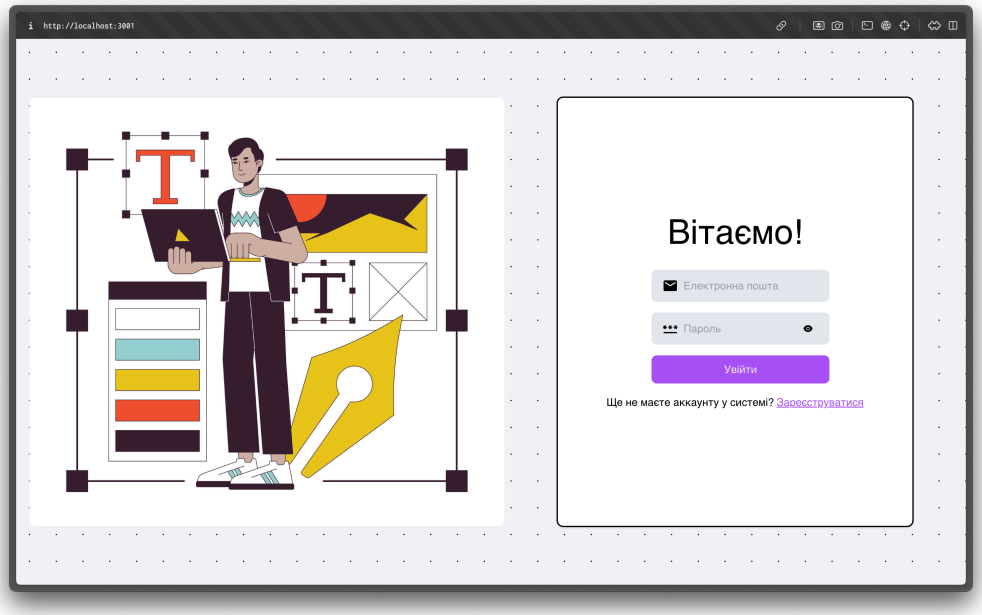


Рисунок 4.1 – Головне вікно аутентицікації в системі.

Справа можемо побачити форму аутентифікації. Напис «Вітаємо» зустрічає нового користувача, який вирішив використовувати даний веб-застосунок. Нижче під написом присутні поля для вводу інформації для ідентифікації в системі, а саме:

- Електронна пошта: поле у яке користувач повинен ввести свою електронну пошту;
- Пароль: поле у яке користувач повинен ввести свій пароль від аккаунту.

З метою забезпечення конфіденційності у полі вводу паролю було запроваджено процес приховування паролю.

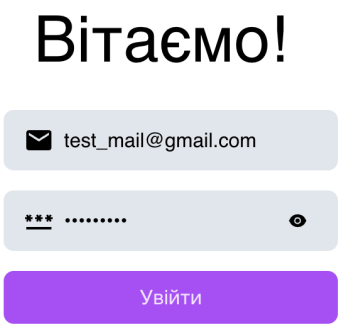


Рисунок 4.2 – Прихований пароль у формі аутентифікації.

Також присутня кнопка «око». При натисканні на цю кнопку, користувач за власним бажанням може показати пароль у полі вводу. Вона працює в обидві сторони.

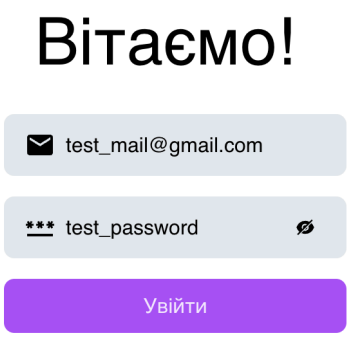


Рисунок 4.3 – Пароль після натискання на кнопку «Око».

Якщо користувач не був зареєстрований в системі до цього, то він має змогу це зробити, натиснувши на кнопку «Зареєструватися». Після цього користувача зустріне форма реєстрації.

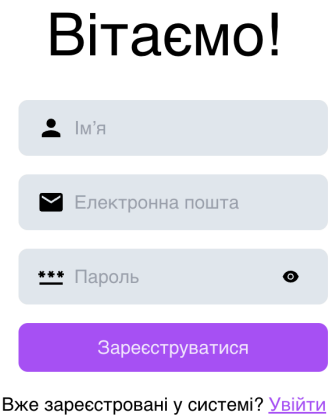


Рисунок 4.4 – Форма реєстрації в системі.

Так як ми вирішили уявити, що ми новий користувач, то давайте зареєструємось в системі з тестовими даними:

- Ім'я – test1;
- Електронна пошта – test1@gmail.com;
- Пароль – test_password;

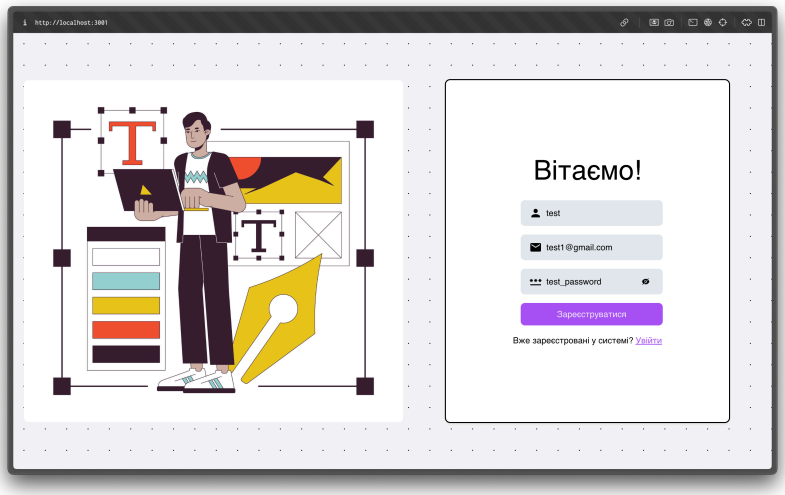


Рисунок 4.5 – Процес реєстрації з тестовими даними.

Результат реєстрації можемо подивитись у командному рядку де був запущений серверний застосунок.

```
backend_1  | Created user:
backend_1  |       name: test1
backend_1  |       email: test1@gmail.com
backend_1  |       hashed password: $2b$10$1dpR6pNBwFGtTYUw6cPr9uFUa0QgMHmU5i1fL
KQni7EmZdltno.E.
```

Рисунок 4.6 – Результат реєстрації.

А також у браузерних інструментах для розробників ми можемо побачити що ідентифікатор користувача та його токен було збережно для подальшої роботи з сервером.

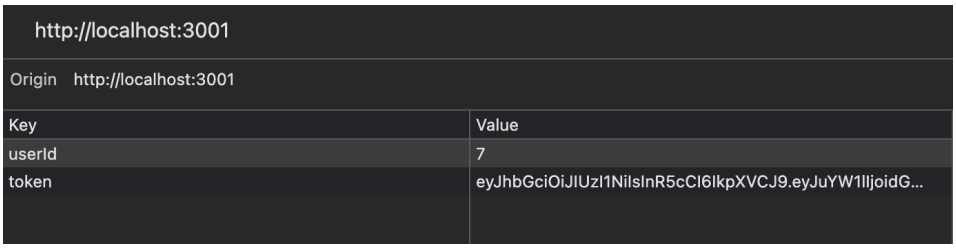


Рисунок 4.7 – Збережений ідентифікатор та токен на клієнті

Після успішної реєстрації, користувача було автоматично переведено до вікна з проектами.

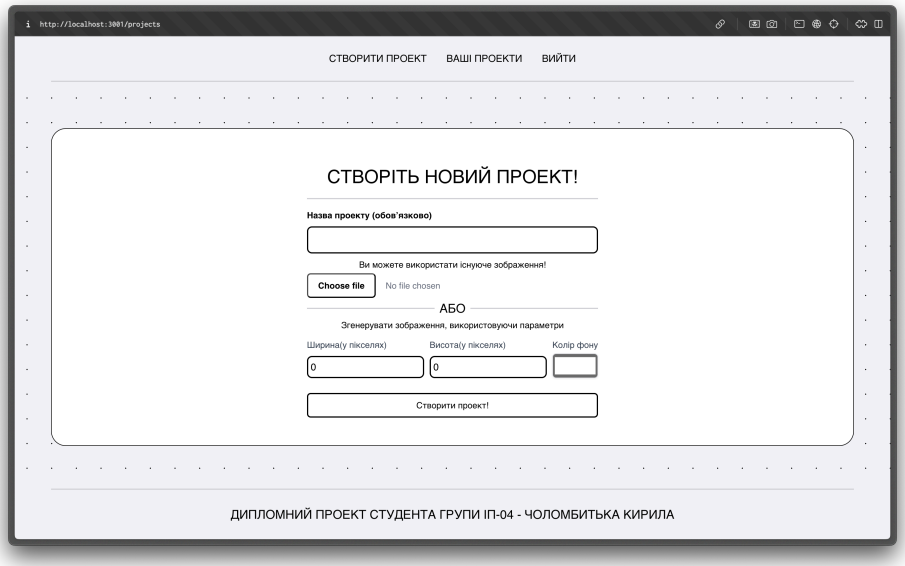


Рисунок 4.6 – Вікно з проектами

У цьому вікні ми одразу бачимо форму створення нового проекту. Ця форма містить такі поля, як:

- Назва проєкту: поле для введення назви проекту для ідентифікації в базі даних;
- Файл: поле для обрання файлу зображення з метою його редагування;

- Ширина: поле для введення ширини згенерованого зображення у пікселях;
- Висота: поле для введення висоти згенерованого зображення у пікселях;
- Колір фону: поле для обрання фонового кольору при генерації зображення;
- Кнопка “Створити проект»: кнопка при натисканні якої створиться новий проект у базі.

Окрім цієї форми ми можемо побачити зверху меню, яке містить вікно для створення проекту, вікно зі списком існуючих ваших проектів та кнопка для виходу з системи. Було створено тестовий проект, після чого одразу відкрилось вікно для редагування згенерованого зображення.

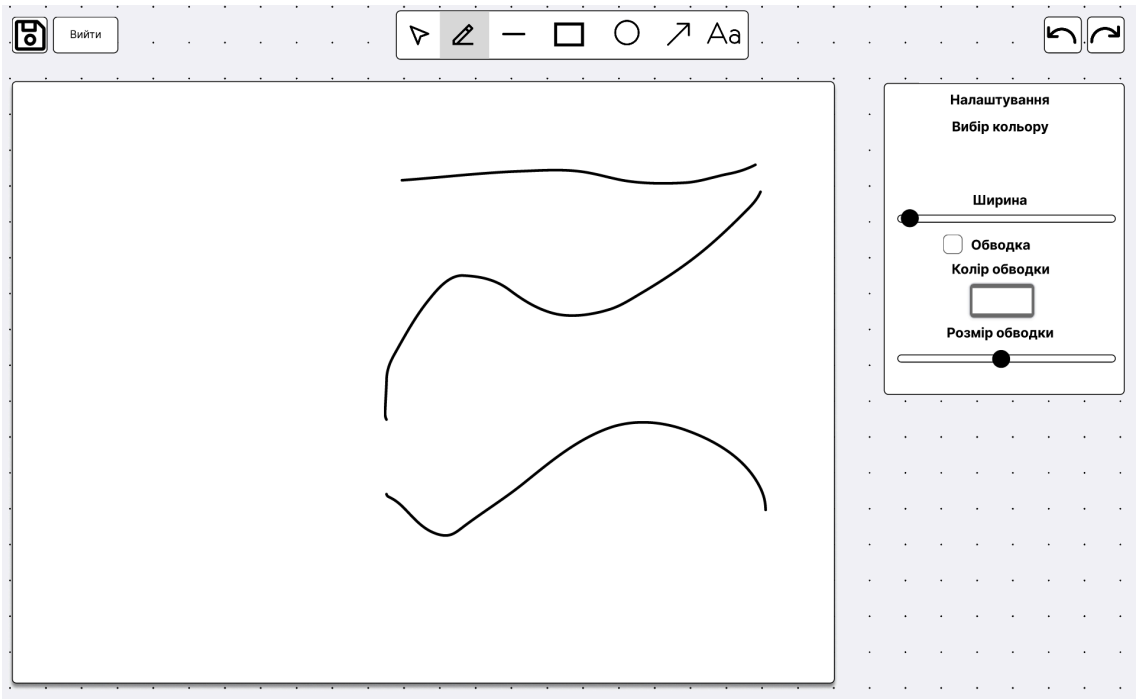


Рисунок 4.7 – Вікно редагування згенерованого зображення

У цьому вікні ми можемо побачити одразу поле з зображенням, яке ми редагуємо. Справа від цього поля є блок з налаштуваннями поточного

інструменті(на прикладі з рисунку – олівця). Зліва зверху було реалізовано дві кнопки:

- Зберегти: кнопка яка відповідає за збереження на сервері зображення;
- Вийти: кнопка яка закінчує виконання поточного вікна.

Посередині ми можемо побачити меню інструментів які можна використати, а саме:

- Курсор: цей інструмент присутній для навігації по зображенню;
- Олівець: дає можливість створювати вільні лінії на зображенні;
- Пряма лінія: інструмент для створення прямої лінії на зображенні;
- Прямокутник: інструмент для створення прямокутника на зображенні;
- Еліпс: завдяки цьому інструменту можемо створити еліпс;
- Стрілка: дає можливість створення стрілки;
- Текст: можемо створити та розмістити власний текст на зображенні.
- Повернути зміну: ця кнопка прибирає останню зміну.
- Повернути прибирання зміни: ця кнопка повертає зміну яку було прибрано в разі натискання кнопки «Повернути зміну»

ВИСНОВОК ДО РОЗДІЛУ 4

У даному розділі було описано процес тестування веб-застосунку імітуючи нового користувача. Був описаний увесь реалізований користувацький інтерфейс, як з ним взаємодіяти та його можливості. Також були перевірені функції редагування та створення зображень. Було перевірено роботу аутентифікації та реєстрації. Основуючись на перевірку, можна прийти до висновку, що програмне забезпечення було правильно реалізовано.

ВИСНОВОК

Під час виконання дипломного проекту було проаналізовано важливість поширення сервісів для редагування і створення зображень та сенс їх використання.

У першому розділі було оглянуто та проаналізовано існуючі рішення. Були виявленні недоліки та переваги веб-застосунків, а саме: Figma, Excalidraw та Canva. Були визначені поняття, виділені критерії та функціонал, на які було звернуто увагу під час розробки нового рішення. Основними критеріями стали: зручний користувацький інтерфейс, більш загальний та простий функціонал, наявні заходи конфіденційності даних та українська локалізація.

У наступному розділі було обрано архітектуру веб-застосунку, а саме клієнт-сервер. Після цього було описано технології які використовувались в процесі розробки програмного забезпечення. Мовою програмування став TypeScript, а середовище запуску - NodeJS. На серверній частині було використано фреймворки NestJS, TypeORM, а також база даних PostgreSQL. Для розробки клієнтської частини було обрано Vite, HTML, Axios, CSS, React, TailwindCSS та Konva.

У третьому розділі було поєтапно описано увесь процес розробки. Спочатку було налаштовано середу для розробки програмного забезпечення, а потім створено конфігурації як для серверної частини, так і для клієнтської. Під час процесу налаштування серверної частини було встановлено та сконфігуровано фреймворки та залежності для розробки. Також було налаштовано docker та docker-compose для усунення проблем з залежностями та управлінням серверними сервісами. Після цього було розроблено необхідне API для роботи з базою та аутентифікацією. Під час розробки клієнтської частини було налаштовано середовище Vite та React для побудування

користувацького інтерфейсу. У цьому ще допоміг TailwindCSS та можливості цього фреймворку. Після цього було побудовано та реалізовано компоненту структуру на клієнтській частині веб-застосунку, а також обробники для надсилання запитів на серверну частину.

У четвертому розділі було перевірено систему від нового користувача. Був описаний користувацький інтерфейс, взаємодія з ним та перевірено його працездатність. Також було перевірено роботу створення, а також редагування зображень, використовуючи реалізовані інструменти.

Основуючись на виконану роботу можна прийти до висновку, що нове програмне забезпечення було реалізовано правильно та задовольняє поставленим критерії та вимоги.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Застосунок - тлумачний словник української мови. [Електронний ресурс]. Режим доступу до ресурсу: https://slovnyk.ua/index.php/index.php?swrd=застосунок.
2. Веб-застосунок - тлумачний словник української мови. [Електронний ресурс]. Режим доступу до ресурсу: https://slovnyk.ua/index.php?swrd=веб-застосунок.
3. Українська бібліотечна енциклопедія. [Електронний ресурс]. Режим доступу до ресурсу: https://ube.nlu.org.ua/article/Інтерфейс користувача.
4. Canva. [Електронний ресурс]. Режим доступу до ресурсу: https://www.canva.com/.
5. Excalidraw – Collaborative whiteboarding made easy. [Електронний ресурс] Режим доступу до ресурсу: https://excalidraw.com/.
6. Figma. [Електронний ресурс]. Режим доступу до ресурсу: https://www.figma.com/.
7. Рейтинг мов програмування - DOU. [Електронний ресурс]. Режим доступу до ресурсу: https://dou.ua/lenta/articles/language-rating-2024/
8. TypeScript. [Електронний ресурс]. Режим доступу до ресурсу: https://www.typescriptlang.org/
9. NodeJS. [Електронний ресурс]. Режим доступу до ресурсу: https://nodejs.org/en
10. NPM. [Електронний ресурс]. Режим доступу до ресурсу: https://www.npmjs.com/
11. HTML - Mozilla Developer Network. [Електронний ресурс]. Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics

12. CSS - Mozilla Developer Network. [Електронний ресурс]. Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics
13. Tailwind. [Електронний ресурс]. Режим доступу до ресурсу: https://tailwindcss.com/
14. React. [Електронний ресурс]. Режим доступу до ресурсу: https://react.dev/
15. Axios. [Електронний ресурс]. Режим доступу до ресурсу: https://axios-http.com/
16. Konva. [Електронний ресурс]. Режим доступу до ресурсу: https://konvajs.org/
17. Vite. [Електронний ресурс]. Режим доступу до ресурсу: https://vitejs.dev/
18. NestJS. [Електронний ресурс]. Режим доступу до ресурсу: https://nestjs.com/
19. PostgreSQL. [Електронний ресурс]. Режим доступу до ресурсу: https://www.postgresql.org/
20. TypeORM. [Електронний ресурс]. Режим доступу до ресурсу: https://typeorm.io/
21. Docker. [Електронний ресурс]. Режим доступу до ресурсу: https://www.docker.com/

ДОДАТОК 1

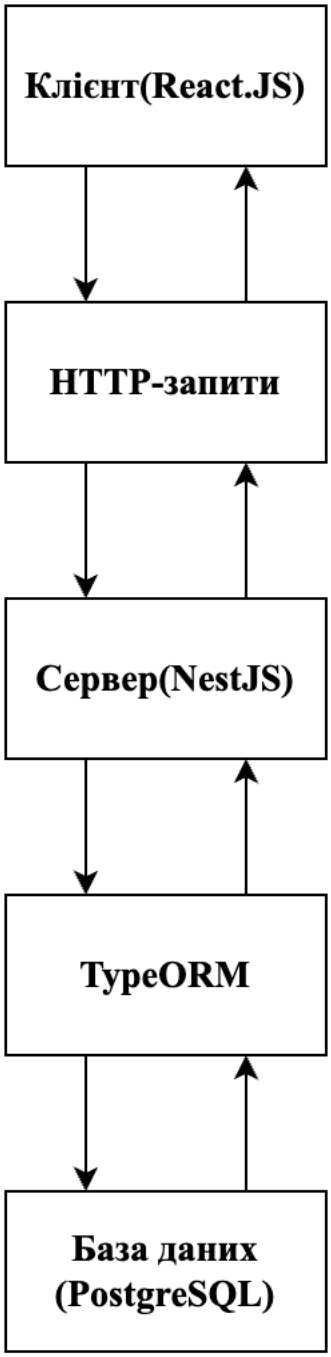
Веб-застосунок для створення та редагування зображень

Компоненти системи (структурна схема)

ІАЛЦ.467100.004 Д1

Аркушів 1

Київ 2024 р



					ІАЛЦ.467100.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Чоломбитько К.О.			**Веб-застосунок для створення та редагування зображень** Компоненти системи (структурна схема)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Алещенко О.В.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІП-04	
Н. Контр.		Волокита А.М.					
Затвердив							

ДОДАТОК 2

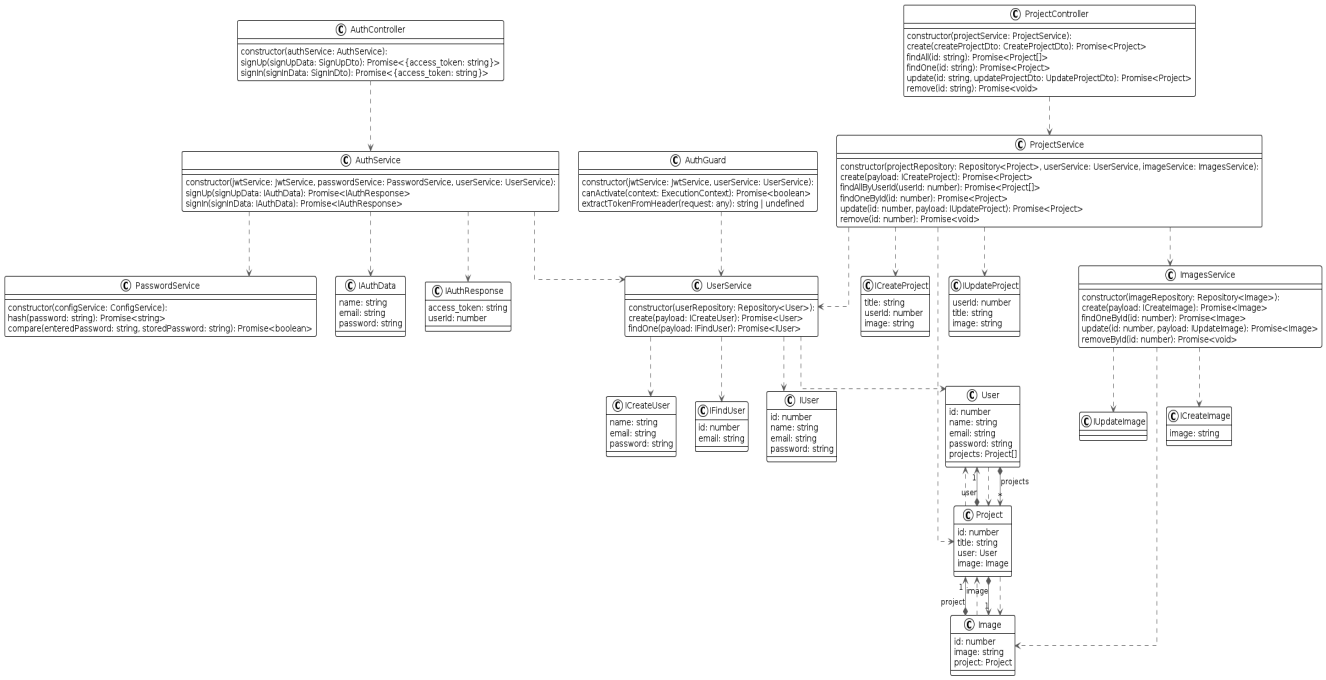
Веб-застосунок для створення та редагування зображень

Діаграма класів (функціональна схема)

ІАЛЦ.467100.005 Д2

Аркушів 1

Київ 2024 р



					ІАЛЦ.467100.005 Д2	
		№ докум.	Підпис	Дата		
Розробив		Чоломбитько К.О.			**Веб-застосунок для створення та редагування зображень**	Літ. / Аркуш / Аркушів
Перевірив		Алешенко В.О.				1 / 1
					Діаграма класів (функціональна схема)	КПІ ім. Ігоря Сікорського, ФІОТ, ІП-04
Н. Контр.		Волокита А.М.				
Затвердив						

ДОДАТОК 3

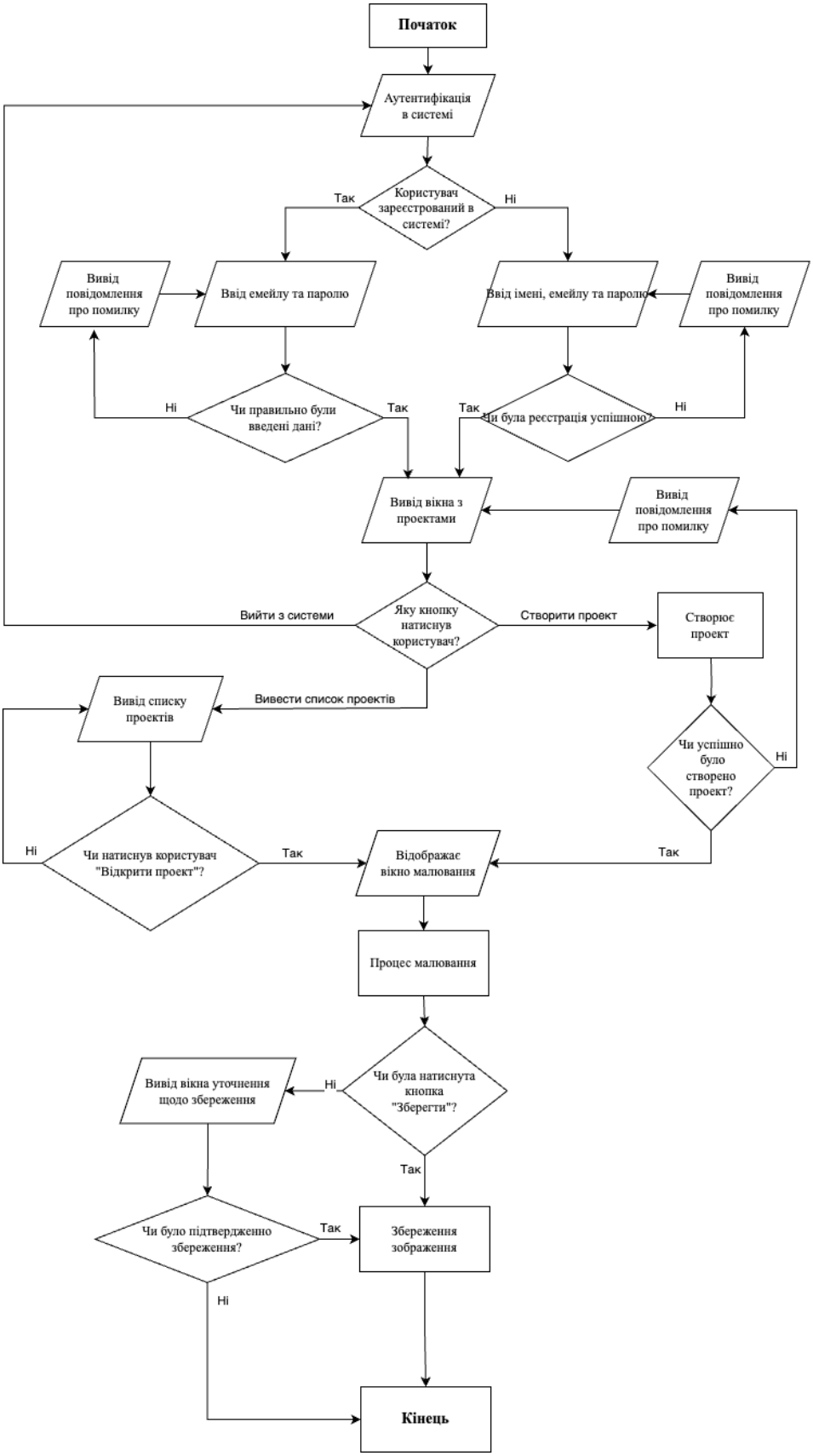
Веб-застосунок для створення та редагування зображень

Алгоритм дій системи (принципова схема)

ІАЛЦ.467100.006 Д3

Аркушів 1

Київ 2024 р



					ІАЛЦ.467100.006 Д3	
		№ докум.	Підпис	Дата		
Розробив		Чоломбитько К.О.			**Веб-застосунок для створення та редагування зображень**	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Алещенко О.В.			Алгоритм дій системи (принципова схема)	КПІ ім. Ігоря Сікорського, ФІОТ, ІП-04
Н. Контр.		Волокита А.М.				
Затвердив						

ДОДАТОК 4

Веб-застосунок для створення та редагування зображень

Текст програмного коду

ІАЛЦ.467100.007 Д4

Аркушів 25

Київ 2024 р

Main.ts

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { ValidationPipe } from '@nestjs/common';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.useGlobalPipes(new ValidationPipe());
  app.enableCors();
  await app.listen(3000);
}
bootstrap();
```

DatabaseModule.ts

```
import { Module } from '@nestjs/common';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from './user/entities/user.entity';
import { Project } from './project/entities/project.entity';
import { Image } from './images/entities/image.entity';
@Module({
  imports: [
    TypeOrmModule.forRootAsync({
      imports: [ConfigModule],
      inject: [ConfigService],
      useFactory: (configService: ConfigService) => ({
        type: 'postgres',
        host: configService.get('HOST'),
        port: configService.get('DB_PORT'),
        username: configService.get('USERNAME'),
        password: configService.get('PASSWORD'),
        database: configService.get('DB_NAME'),
        entities: [User, Project, Image],
        synchronize: true,
      }),
    }),
  ],
})
export class DatabaseModule {}
```

AppModule.ts

```
import { Module } from '@nestjs/common';
import { UserModule } from './user/user.module';
import { ConfigModule } from '@nestjs/config';
import { DatabaseModule } from './database.module';
import { AuthModule } from './auth/auth.module';
import { ProjectModule } from './project/project.module';
import { ImagesModule } from './images/images.module';
import { JwtModule } from '@nestjs/jwt';
import * as dotenv from 'dotenv';
dotenv.config();

@Module({
  imports: [
    ConfigModule.forRoot({ isGlobal: true }),
    JwtModule.register({
      global: true,
      secret: process.env.JWT_SECRET,
    }),
    DatabaseModule,
    UserModule,
    AuthModule,
    ProjectModule,
    ImagesModule,
  ],
})
export class AppModule {}
```

AuthService.ts

```
import { Injectable } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { PasswordService } from './password.service';
import { UserService } from 'src/user/user.service';
import { IAuthData } from '../interfaces/auth-data.interface';
import { UserAlreadyExistsException } from '../exceptions/user-already-exists.exception';
import { UserNotFoundException } from '../exceptions/user-not-found.exception';
import { IncorrectPasswordException } from '../exceptions/incorrect-password.exception';
import { IAuthResponse } from '../interfaces/auth-response.interface';

@Injectable()
export class AuthService {
  constructor(
    private jwtService: JwtService,
    private passwordService: PasswordService,
    private userService: UserService,
  ) {}

  async signUp(signUpData: IAuthData): Promise<IAuthResponse> {
    const user = await this.userService.findOne({
      email: signUpData.email,
    });

    if (user) throw new UserAlreadyExistsException();

    const hash = await this.passwordService.hash(signUpData.password);

    const newUser = await this.userService.create({
      name: signUpData.name,
      email: signUpData.email,
      password: hash,
    });

    const payload = { name: newUser.name, id: newUser.id };

    const token = await this.jwtService.signAsync(payload);

    return { access_token: token, userId: newUser.id };
  }

  async signIn(signInData: IAuthData): Promise<IAuthResponse> {
```

```
    const user = await this.userService.findOne({
      email: signInData.email,
    });

    if (!user) throw new UserNotFoundException();

    const isPasswordCorrect = await this.passwordService.compare(
      signInData.password,
      user.password,
    );

    if (!isPasswordCorrect) throw new IncorrectPasswordException();

    const payload = { name: user.name, id: user.id };

    const token = await this.jwtService.signAsync(payload);

    return { access_token: token, userId: user.id };
  }
}
```

PasswordService.ts

```
import { Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import * as bcrypt from 'bcrypt';

@Injectable()
export class PasswordService {
  constructor(private configService: ConfigService) {}
  async hash(password: string): Promise<string> {
    const salt = this.configService.get<string>('SALT');

    return bcrypt.hash(password, salt);
  }

  async compare(
    enteredPassword: string,
    storedPassword: string,
  ): Promise<boolean> {
    return bcrypt.compare(enteredPassword, storedPassword);
  }
}
```

AuthGuard.ts

```
import { CanActivate, ExecutionContext, Injectable } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { UserService } from 'src/user/user.service';
import { AuthTokenNotFoundException } from '../exceptions/no-auth-token.exception';
import { InvalidTokenException } from '../exceptions/invalid-token-exception';
import { UserNotFoundException } from '../exceptions/user-not-found.exception';

@Injectable()
export class AuthGuard implements CanActivate {
  constructor(
    private jwtService: JwtService,
    private userService: UserService,
  ) {}
  async canActivate(context: ExecutionContext): Promise<boolean> {
    const request = context.switchToHttp().getRequest();
    const token = this.extractTokenFromHeader(request);

    if (!token) throw new AuthTokenNotFoundException();

    const payload = await this.jwtService.verifyAsync(token);

    if (!payload) throw new InvalidTokenException();

    const user = await this.userService.findOne({ email: payload.email });

    if (!user) throw new UserNotFoundException();

    return true;
  }

  private extractTokenFromHeader(request): string | undefined {
    const [type, token] = request.headers.authorization?.split(' ') ?? [];
    return type === 'Bearer' ? token : undefined;
  }
}
```

AuthModule.ts

```
import { Module } from '@nestjs/common';
import { AuthService } from './services/auth.service';
```

```
import { AuthController } from './auth.controller';
import { PasswordService } from './services/password.service';
import { UserModule } from 'src/user/user.module';

@Module({
  imports: [UserModule],
  providers: [AuthService, PasswordService],
  controllers: [AuthController],
})
export class AuthModule {}
```

ImagesService.ts

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { Image } from './entities/image.entity';
import { ICreateImage } from './interfaces/create-image.interface';
import { IUpdateImage } from './interfaces/update-image.interface';

@Injectable()
export class ImagesService {
  constructor(
    @InjectRepository(Image)
    private imageRepository: Repository<Image>,
  ) {}

  async create(payload: ICreateImage): Promise<Image> {
    const image = this.imageRepository.create(payload);
    return this.imageRepository.save(image);
  }

  async findOneById(id: number): Promise<Image> {
    return await this.imageRepository.findOne({ where: { id } });
  }

  async update(id: number, payload: IUpdateImage): Promise<Image> {
    await this.imageRepository.update(id, payload);
    return this.findOneById(id);
  }

  async removeById(id: number): Promise<void> {
    const image = await this.findOneById(id);
```

```
    await this.imageRepository.remove(image);
  }
}
```

CreateImageInterface.ts

```
export interface ICreateImage {
  image: string;
}
```

ImagesModule.ts

```
import { Module } from '@nestjs/common';
import { ImagesService } from './images.service';
import { Image } from './entities/image.entity';
import { TypeOrmModule } from '@nestjs/typeorm';

@Module({
  imports: [TypeOrmModule.forFeature([Image])],
  providers: [ImagesService],
  exports: [ImagesService],
})
export class ImagesModule {}
```

ProjectsService.ts

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { Project } from './entities/project.entity';
import { ImagesService } from 'src/images/images.service';
import { ICreateProject } from './interface/create-project.interface';
import { UserService } from 'src/user/user.service';
import { IUpdateProject } from './interface/update-project.interface';

@Injectable()
export class ProjectService {
  constructor(
    @InjectRepository(Project)
    private projectRepository: Repository<Project>,
    private userService: UserService,
    private imageService: ImagesService,
```

```
) {}

async create(payload: ICreateProject): Promise<Project> {
  const user = await this.userService.findOne({ id: payload.userId });
  if (!user) {
    throw new NotFoundException('User not found');
  }

  const image = await this.imageService.create({ image: payload.image });

  const project = await this.projectRepository.create({
    title: payload.title,
    user,
    image,
  });
  return await this.projectRepository.save(project);
}

async findAllByUserId(userId: number): Promise<Project[]> {
  const projects = await this.projectRepository
    .createQueryBuilder('project')
    .leftJoinAndSelect('project.user', 'user')
    .leftJoinAndSelect('project.image', 'image')
    .where('user.id = :userId', { userId })
    .getMany();

  return projects;
}

async findOneById(id: number): Promise<Project> {
  const project = await this.projectRepository
    .createQueryBuilder('project')
    .leftJoinAndSelect('project.user', 'user')
    .leftJoinAndSelect('project.image', 'image')
    .where('project.id = :id', { id })
    .getOne();

  if (!project) {
    throw new NotFoundException(`Project #${id} not found`);
  }
  return project;
}
```

```
  async update(id: number, payload: IUpdateProject): Promise<Project> {
    const project = await this.findOneById(id);

    if (payload.image) {
      await this.imageService.update(project.image.id, {
        image: payload.image,
      });
      project.image.image = payload.image;
    }

    if (payload.title !== undefined) {
      project.title = payload.title;
    }

    return this.projectRepository.save(project);
  }

  async remove(id: number): Promise<void> {
    const project = await this.findOneById(id);
    await this.projectRepository.remove(project);
  }
}
```

ProjectController.ts

```
import {
  Controller,
  Get,
  Post,
  Body,
  Patch,
  Param,
  Delete,
  UseGuards,
} from '@nestjs/common';
import { ProjectService } from './project.service';
import { CreateProjectDto } from './dto/create-project.dto';
import { UpdateProjectDto } from './dto/update-project.dto';
import { AuthGuard } from 'src/auth/guards/auth.guard';

@UseGuards(AuthGuard)
@Controller('project')
export class ProjectController {
```

```
  constructor(private readonly projectService: ProjectService) {}

  @Post()
  create(@Body() createProjectDto: CreateProjectDto) {
    return this.projectService.create(createProjectDto);
  }

  @Get('/findByUserId/:id')
  findAll(@Param('id') id: string) {
    return this.projectService.findAllByUserId(+id);
  }

  @Get(':id')
  findOne(@Param('id') id: string) {
    return this.projectService.findOneById(+id);
  }

  @Patch(':id')
  update(@Param('id') id: string, @Body() updateProjectDto: UpdateProjectDto) {
    return this.projectService.update(+id, updateProjectDto);
  }

  @Delete(':id')
  remove(@Param('id') id: string) {
    return this.projectService.remove(+id);
  }
}
```

CreateProjectInterface.ts

```
export interface ICreateProject {
  title: string;
  userId: number;
  image: string;
}
```

CreateProjectDto.ts

```
import { IsInt, IsNotEmpty, IsString } from 'class-validator';

export class CreateProjectDto {
  @IsNotEmpty()
  @IsInt()
```

```
  userId: number;

  @IsNotEmpty()
  @IsString()
  title: string;

  @IsNotEmpty()
  @IsString()
  image: string;
}
```

ProjectModule.ts

```
import { Module } from '@nestjs/common';
import { ProjectService } from './project.service';
import { ProjectController } from './project.controller';
import { TypeOrmModule } from '@nestjs/typeorm';
import { Project } from './entities/project.entity';
import { ImagesModule } from 'src/images/images.module';
import { UserModule } from 'src/user/user.module';

@Module({
  imports: [TypeOrmModule.forFeature([Project]), ImagesModule, UserModule],
  controllers: [ProjectController],
  providers: [ProjectService],
})
export class ProjectModule {}
```

App.tsx

```
import { FC } from "react";
import React from "react";
import AuthPage from "./components/Auth/AuthPage";
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import Project from "./components/Project/Project";

const App: FC = () => {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<AuthPage/>}/>
        <Route path="/projects" element={<Project/>}/>
      </Routes>
```

```
    </Router>
  );
}

export default App;
```

auth.ts

```
import axios from "axios";

const BACKEND_AUTH_URL = "http://localhost:3000/auth/"

interface ISignUp {
	name: string;
	email: string;
	password: string;
}

interface ISignUpResponseData {
	userId: number;
	access_token: string;
}

interface ISignIn {
	email: string;
	password: string;
}

export const signUp = async (payload: ISignUp) => {
	try {
		const response = await axios.post(`${BACKEND_AUTH_URL}signUp`, payload)
		const data: ISignUpResponseData = response.data;
		return data;
	}	catch(e) {
		console.error(e.response.data.message);
		return e.response.data.message;
	}
}

export const signIn = async (payload: ISignIn) => {
	try {
		const response = await axios.post(`${BACKEND_AUTH_URL}signIn`, payload)
		const data = response.data;
```

```
		console.log(data);
		return data;
	}	catch(e) {
		console.error(e.response.data.message);
		return e.response.data.message;
	}
}
```

project.ts

```
import axios from "axios";

const BACKEND_PROJECT_URL = "http://localhost:3000/project/"

interface ICreateProject {
	userId: number;
	title: string;
	image: string;
}

interface IUpdateProject {
	userId: number;
	title?: string;
	image?: string;
}

export const getAllProjectsByUserId = async (access_token: string, userId: number) => {
	try {
		const response = await
axios.get(`${BACKEND_PROJECT_URL}findByUserId/${userId}`,
			{ headers: { Authorization: `Bearer ${access_token}` }}
		);
		const data = response.data;
		return data;
	}	catch(e) {
		console.error(e.response.data.message);
		return e.response.data.message;
	}
}

export const getProjectById = async (access_token: string, project_id: number) => {
	try {
		const response = await axios.get(`${BACKEND_PROJECT_URL}`,
```

```
                { headers: { Authorization: `Bearer ${access_token}` }}
            );
            const data = response.data;
            return data;
        }   catch(e) {
            console.error(e.response.data.message);
            return e.response.data.message;
        }
}

export const createProject = async (payload: ICreateProject, access_token: string) => {
        try {
            const response = await axios.post(`${BACKEND_PROJECT_URL}`,
                payload,
                { headers: { Authorization: `Bearer ${access_token}` }}
            );
            const data = response.data;
            return data;
        }   catch(e) {
            console.error(e.response.data.message);
            return e.response.data.message;
        }
}

export const updateProjectById = async (
        payload: IUpdateProject,
        access_token: string,
        project_id: number
) => {
        try {
            const response = await axios.patch(`${BACKEND_PROJECT_URL}/${project_id}`,
                payload,
                { headers: { Authorization: `Bearer ${access_token}` }}
            )
            const data = response.data;
            console.log(data);
            return data;
        }   catch(e) {
            console.error(e.response.data.message);
            return e.response.data.message;
        }
}
```

```
export const deleteProjectById = async (project_id: number, access_token: string) => {
	try {
		const response = await axios.delete(`${BACKEND_PROJECT_URL}/${project_id}`,
			{ headers: { Authorization: `Bearer ${access_token}` }}
		)
		const data = response.data;
		console.log(data);
		return data;
	}	catch(e) {
		console.error(e.response.data.message);
		return e.response.data.message;
	}
}
```

Auth.tsx

```
import React from "react";
import { FC, useState } from "react";
import LoginForm from "./Login";
import RegisterForm from "./Register";

const authTypeChange = {
	"Login": "Register",
	"Register": "Login"
}

const Auth: FC = () => {
	const [authType, setAuthType] = useState("Login");

	const handleAuthType = (event) => {
		event.preventDefault();
		setAuthType(authTypeChange[authType]);
	}

	return (
		<div className="bg-white outline outline-black outline-2 rounded-lg shadow-md
h-full mx-20">
			<div className="h-full w-full flex flex-col items-center justify-
center">
				<h1 className="text-black mb-5">Вітаємо!</h1>
				{authType === "Login" ? <LoginForm/> : <RegisterForm/>}
				<div className="flex flex-row mt-2">
					<div className="text-black">
```

```
                                        {authType === "Login" ?
                                                "Ще не маєте аккаунту у системі?" :
                                                "Вже зареєстровані у системі?"
                                        }
                                </div>
                                <div
                                        className="cursor-pointer text-purple-500 ml-1
underline hover:text-purple-300"
                                        onClick={handleAuthType}
                                >
                                        {authType === "Login" ?
                                                "Зареєструватися" :
                                                "Увійти"
                                        }
                                </div>
                        </div>
                </div>
        </div>
    );
}

export default Auth;
```

Login.tsx

```
import React from "react";
import { FC, useState } from "react";
import { MdEmail, MdPassword } from "react-icons/md";
import { AiFillEye, AiFillEyeInvisible } from "react-icons/ai";
import { signIn } from "../../lib/auth";
import { useNavigate } from "react-router-dom";

const LoginForm: FC = () => {
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [isVisible, setIsVisible] = useState(false);
    const [error, setError] = useState('');
    const navigate = useNavigate();

    const handleVision = (event) => {
        event.preventDefault();
        setIsVisible(!isVisible);
    }
```

```
	const handleEmail = (event) => {
		event.preventDefault();
		setEmail(event.target.value);
		return;
	}

	const handlePassword = (event) => {
		event.preventDefault();
		setPassword(event.target.value);
		return;
	}

	const handleLogin = async (event) => {
		event.preventDefault();
		try {
			const data = await signIn({ email, password });
			console.log(data);
			if(typeof data === 'string') setError(data);
			else {
				localStorage.setItem("token", data.access_token);
				localStorage.setItem("userId", data.userId);
				setError('');
				navigate('/projects');
			}
		} catch(e) {
		}
	}

	return (
		<div>
			<form onSubmit={handleLogin} className="flex flex-col w-full">
				<div className="flex h-12 bg-gray-200 m-2 p-2 rounded-lg w-
full">
					<MdEmail className="size-10 self-center p-2"
color="black"/>
					<input
						value={email}
						onChange={handleEmail}
						className="w-fit bg-transparent text-black"
						placeholder="Електронна пошта">
					</input>
				</div>
```

```
                        <div className="flex h-12 bg-gray-200 m-2 p-2 rounded-lg w-
full">
                                <MdPassword className="size-10 self-center p-2"
color="black"/>
                                <input
                                        type={isVisible ? "text" : "password"}
                                        className="w-fit bg-transparent text-black"
placeholder="Пароль"
                                        value={password}
                                        onChange={handlePassword}
                                >
                                        </input>
                                        {isVisible ?
                                        <AiFillEyeInvisible
                                                className="cursor-pointer self-center
                                        hover:fill-gray-500"
                                                color="black"
                                                onClick={handleVision}
                                        /> :
                                        <AiFillEye
                                                className="cursor-pointer self-center
                                                hover:fill-gray-500"
                                         color="black"
                                         onClick={handleVision}
                                        />}
                        </div>
                        <div className="text-red-600 self-center">{error ? error :
''}</div>
                        <button type="submit" className="bg-purple-500 m-2 p-2 hover:bg-
purple-400 w-full">Увійти</button>
                    </form>
             </div>
       );
}

export default LoginForm;
```

Register.tsx

```
import React from "react";
import { FC, useState } from "react";
import { AiFillEyeInvisible, AiFillEye } from "react-icons/ai";
import { MdEmail, MdPassword, MdPerson } from "react-icons/md";
```

```
import { signUp } from "../../lib/auth";
import { useNavigate } from "react-router-dom";

const RegisterForm: FC = () => {
	const [name, setName] = useState('');
	const [email, setEmail] = useState('');
	const [password, setPassword] = useState('');
	const [error, setError] = useState('');
	const [isVisible, setIsVisible] = useState(false);
	const navigate = useNavigate();

	const handleVision = (event) => {
		event.preventDefault();
		setIsVisible(!isVisible);
	}

	const handleName = (event) => {
		event.preventDefault();
		setName(event.target.value);
		return;
	}

	const handleEmail = (event) => {
		event.preventDefault();
		setEmail(event.target.value);
		return;
	}

	const handlePassword = (event) => {
		event.preventDefault();
		setPassword(event.target.value);
		return;
	}

	const handleRegister = async (event) => {
		event.preventDefault();
		try {
			const data = await signUp({ name, email, password });
			if(typeof data === 'string') setError(data);
			else {
				localStorage.setItem("token", data.access_token);
				localStorage.setItem("userId", data.userId);
				setError('');
```

```
                        navigate('/projects');
                    }
                } catch(e) {
                }
        }

        return (
                <div>
                    <form onSubmit={handleRegister} className="flex flex-col">
                        <div className="flex h-12 bg-gray-200 m-2 p-2 rounded-lg w-
full">
                            <MdPerson className="size-10 self-center p-2"
color="black"/>
                            <input
                                className="w-fit bg-transparent text-black"
                                placeholder="Ім’я"
                                value={name}
                                onChange={handleName}
                            >
                            </input>
                        </div>
                        <div className="flex h-12 bg-gray-200 m-2 p-2 rounded-lg w-
full">
                            <MdEmail className="size-10 self-center p-2"
color="black"/>
                            <input
                                className="w-fit bg-transparent text-black"
                                placeholder="Електронна пошта"
                                value={email}
                                onChange={handleEmail}
                            >
                            </input>
                        </div>
                        <div className="flex h-12 bg-gray-200 m-2 p-2 rounded-lg w-
full">
                            <MdPassword className="size-10 self-center p-2"
color="black"/>
                            <input
                                type={isVisible ? "text" : "password"}
                                className="w-fit bg-transparent text-black"
                                placeholder="Пароль"
                                value={password}
                                onChange={handlePassword}
```

```
                                    >
                                        </input>{isVisible ?
                                        <AiFillEyeInvisible
                                            className="cursor-pointer self-center
hover:fill-gray-500"
                                            color="black"
                                            onClick={handleVision}
                                        /> :
                                        <AiFillEye
                                            className="cursor-pointer self-center
                                            hover:fill-gray-500"
                                            color="black"
                                            onClick={handleVision}
                                        />}
                                </div>
                                <div className="text-red-600 self-center">{error ? error :
''}</div>
                                <button type="submit" className="bg-purple-500 m-2 p-2 hover:bg-
purple-400 w-full">Зареєструватися</button>
                            </form>
                    </div>
            );
}

export default RegisterForm;
```

Project.tsx

```
import React, { FC, useEffect, useState } from "react";
import { getAllProjectsByUserId } from "../../lib/project";
import ProjectList from "./ProjectList";
import ProjectCreate from "./ProjectCreate";
import { useNavigate } from 'react-router-dom';

const Project: FC = () => {
  const [projects, setProjects] = useState([]);
  const [currentView, setCurrentView] = useState("create");
  const navigate = useNavigate();

  useEffect(() => {
    fetchProjects();
  }, []);
```

```
const fetchProjects = async () => {
  try {
    const userId = localStorage.getItem("userId")!;
    const token = localStorage.getItem("token")!;
    const response = await getAllProjectsByUserId(token, +userId);
    setProjects(response);
    console.log(projects);
  } catch (e) {
    console.error(e);
  }
};

const handleLogOut = async () => {
  localStorage.removeItem("userId");
  localStorage.removeItem("token");
  navigate("/");
}

return (
  <div className="flex flex-col justify-between items-center min-h-screen text-black">
    <div className="bg-gray-100 w-screen h-20 flex flex-col items-center">
      <div className="flex justify-center my-4">
        <button
          className={
            `relative focus:outline-none focus:ring-0 focus:border-none
            hover:outline-none hover:ring-0 hover:border-none
            outline-none ring-0 border-none
            px-4 py-2 text-black bg-transparent group`
          }
          onClick={() => setCurrentView("create")}
        >
          СТВОРИТИ ПРОЕКТ
          <span className={
            `absolute bottom-0 left-0 w-full h-0.5 bg-black
            transform scale-x-0 transition-transform duration-300
            ease-in-out group-hover:scale-x-100`
          }></span>
        </button>
        <button
          className={
            `relative focus:outline-none focus:ring-0 focus:border-none
            hover:outline-none hover:ring-0 hover:border-none
            outline-none ring-0 border-none
```

```
        px-4 py-2 text-black bg-transparent group`
      }
      title={projects.length === 0 ? "У вас немає наразі існуючих проектів" : ""}
      onClick={() => setCurrentView("list")}
      disabled={projects.length === 0}
    >
      ВАШІ ПРОЕКТИ
      <span className={
        `absolute bottom-0 left-0 w-full h-0.5 bg-black
        transform scale-x-0 transition-transform duration-300
        ease-in-out group-hover:scale-x-100`
      }>
      </span>
    </button>
    <button
      className={
        `relative focus:outline-none focus:ring-0 focus:border-none
        hover:outline-none hover:ring-0 hover:border-none
        outline-none ring-0 border-none
        px-4 py-2 text-black bg-transparent group`
      }
      onClick={handleLogOut}
    >
      ВИЙТИ
      <span className={
        `absolute bottom-0 left-0 w-full h-0.5 bg-black
        transform scale-x-0 transition-transform duration-300
        ease-in-out group-hover:scale-x-100`
      }>
      </span>
    </button>
  </div>
  <hr className="border-t-2 border-gray-300 w-11/12"/>
</div>
<div className={
  `relative w-11/12 bg-white rounded-3xl overflow-hidden
  h-[calc(100vmin_-_300px)] border border-solid border-black p-10
  flex flex-row justify-center`
 }>
  <div
    className={`absolute w-full transition-transform duration-500 transform ${
      currentView === "create" ? "translate-x-0" : "-translate-x-full"
    }`}
```

```
        >
          <ProjectCreate />
        </div>
        <div
          className={`absolute w-full transition-transform duration-500 transform ${
            currentView === "list" ? "translate-x-0" : "translate-x-full"
          }`}
        >
          <ProjectList project_list={projects} />
        </div>
      </div>
      <div className="w-screen flex flex-col items-center">
        <hr className="border-t-2 border-gray-300 w-11/12"/>
        <div className="bg-gray-100 w-screen h-20 text-center flex items-center justify-
center">
          <h2 className="text-xl text-black">ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТА ГРУПИ ІП-04 -
ЧОЛОМБИТЬКА КИРИЛА</h2>
        </div>
      </div>
    </div>
  );
}

export default Project;
```

ProjectList.tsx

```
import { FC } from "react";
import React from "react";
import ProjectBlock from "./ProjectBlock";

interface IProject {
       id: number;
       title: string;
       user: object;
       image: object;
}

interface IProjectListProps {
       project_list: Array<IProject>;
};

const ProjectList: FC<IProjectListProps> = (props) => {
```

```
	return (
		<div className="flex flex-row">
			{props.project_list.map((project) => (
				<ProjectBlock
					key={project.id}
					title={project.title}
				/>
			))}
		</div>
	);
}

export default ProjectList;
```