

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

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

Сергій СТИРЕНКО

(підпис)

“ ” _____ 2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Веб-застосунок для управління розробкою проєктів

Виконав : студент 4 курсу, групи ІО-93
(шифр групи)

Руденко Станіслав Олександрович

(прізвище, ім’я, по батькові)

(підпис)

Керівник асистент, Русінов В. В.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Консультант (нормоконтроль) ст.викл., Виноградов Ю. М.

(назва розділу)

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

Рецензент _____

(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)

**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

“Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
Сергій СТИРЕНКО

(підпис)

“ ____ ” _____ 2023 р.

ЗАВДАННЯ

на бакалаврський дипломний проєкт студента

Руденка Станіслава Олександровича

1. Тема проєкту Веб-застосунок для управління розробкою проєктів
керівник проєкту Русінов Володимир Володимирович, асистент, к.т.н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від _____ 2023 року № ____
2. Термін здачі студентом закінченого проєкту «9» червня 2023 року
3. Вихідні дані до проєкту технічна документація, теоретичні дані.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються): Огляд існуючих способів формування веб-сторінок, опис принципів побудови веб-серверів, аналіз етапів розробки веб-додатку для управління розробкою веб-додатків
5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень) алгоритм роботи веб застосунку для управління розробкою проєктів (принципова схема), веб-застосунок діаграма класів (функціональна схема), веб-застосунок для управління розробкою проєктів (структурна схема).

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Виноградов Ю. М.		

7. Дата видачі завдання 10.12.2022

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	<i>Затвердження теми проєкту</i>	10.12.2022-15.12.2022	
2.	<i>Вивчення та аналіз завдання</i>	15.12.2022-15.03.2023	
3.	<i>Розробка архітектури та загальної структури системи</i>	15.03.2023-25.03.2023	
4.	<i>Розробка структур окремих підсистем</i>	25.03.2023-5.04.2023	
5.	<i>Програмна реалізація системи</i>	5.04.2023-15.04.2023	
6.	<i>Оформлення пояснювальної записки</i>	15.05.2023-05.06.2023	
7.	<i>Захист програмного продукту</i>	05.06.2023	
8.	<i>Передзахист</i>	20.06.2023	
9.	<i>Захист</i>	22.06.2023	

Студент-дипломник _____
(підпис)

Керівник проєкту _____
(підпис)

АНОТАЦІЯ

У даній роботі було детально розглянуто підходи до розробки веб-додатків (як клієнтської, так і серверної частини) на прикладі створення додатку для керування розробкою проектів. Були проаналізовані їхні слабкі та сильні сторони. На основі аналізу було вибрані найкращі підходи, які лягли в основу вирішення задачі по створенню додатку. Протягом цієї роботи ми досліджували різні аспекти створення такої програми, включаючи визначення вимог, проектування інтерфейсів користувача, впровадження функціональних можливостей і тестування для забезпечення якості. Під час цієї роботи ми стикалися з проблемами та перешкодами, але ми їх подолали завдяки наполегливості, співпраці та інноваціям. Веб-додаток був розроблений на мові JavaScript.

Ключові слова: веб-додаток, сервер, клієнт, JavaScript.

ANNOTATION

In this project for a Bachelor's Degree the approaches to the development of web applications (both client and server part) were considered in detail, using the example of creating an application for managing the development of projects. Their weaknesses and strengths were analyzed. Based on the analysis, the best approaches were selected, which formed the basis of solving the task of creating an application. Throughout this work, we explored various aspects of building such an application, including defining requirements, designing user interfaces, implementing functionality, and testing for quality assurance. We faced challenges and obstacles during this work, but we overcame them through persistence, collaboration and innovation. The web application was developed in JavaScript.

Keywords: web application, server, client, JavaScript.

справки	Формат	Значення			Найменування	Кіл. листів	№ екземпля	Додаток				
					Документація загальна							
	A4	ІАЛЦ.467800.002 ТЗ			Веб-застосунок	3						
					для управління розробкою							
					проектів							
					Технічне завдання							
	A4	ІАЛЦ.467800.003 ПЗ			Веб-застосунок	80						
					для управління розробкою							
					проектів							
					Пояснювальна записка							
	A4	ІАЛЦ.467800.004 Д1			Веб-застосунок	1						
					для управління розробкою							
					проектів							
					(структурна схема)							
	A4	ІАЛЦ.467800.005 Д2			Веб-застосунок	1						
					для управління розробкою							
					проектів							
					Веб-застосунок діаграма							
					класів							
					(функціональна схема)							
	A4	ІАЛЦ.467800.006 Д3			Веб-застосунок	1						
					для управління розробкою							
					проектів							
					Алгоритм роботи веб							
					застосунку для управління							
					розробкою проектів							
					(принципова схема)							
	A4	ІАЛЦ.467800.007 Д4			Веб-застосунок	24						
					для управління розробкою							
					проектів							
					Текст програмного коду							
					ІАЛЦ.467800.001 ОА							
Зм	Лист	№ докум.	Підп	Дата								
Розроб		Руденко С.О			Веб-застосунок для управління розробкою проектів Опис альбому	Літ.		Аркуш	Аркушів			
Перев		Русінов В.В.						1	1			
						КПІ ім. Ігоря Сікорського ФІОТ ІО-93						

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Веб-застосунок для управління розробкою проєктів»

Київ – 2023

ЗМІСТ

НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ	2
ПІДСТАВИ ДЛЯ РОЗРОБКИ	2
МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ.....	2
ДЖЕРЕЛА РОЗРОБКИ.....	2
ТЕХНІЧНІ ВИМОГИ.....	3
Вимоги до розробленого продукту	3
Вимоги до програмного забезпечення	3
Вимоги до апаратної частини	3
ЕТАПИ РОЗРОБКИ	3

					ІАЛЦ.467800.002 ТЗ			
		№ докум.	Підпис	Дата				
Розробив	Руденко С.О.				Веб-застосунок для управління розробкою проєктів Технічне завдання	Літ.	Аркуш	Аркушів
Перевірив	Русінов В.В.						1	3
Н. Контр.	Виноградов Ю.М.					КПІ ім. Ігоря Сікорського, ФІОТ, ІО-93		
Затвердив								

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку системи для керування розробкою проектів, а також на подальшу підтримку та вдосконалення розробленої системи.

Сфера застосування веб-додатку для управління розробкою проекту є всеосяжною та охоплює різні аспекти управління проектом. Основна мета цієї веб-програми — оптимізувати й оптимізувати процеси розробки проекту, покращити співпрацю та підвищити загальний успіх проекту. Ось кілька ключових сфер застосування: планування та ініціювання проекту, керування завданнями та робочим процесом, співпраця та спілкування розробників, управління часом і розкладом.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка веб-застосунку для управління розробкою проектів, для покращення співпраці розробників програмного забезпечення

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації для виконання цієї роботи є технічна література, а також електронні статті у мережі Інтернет.

					ІАЛЦ.467800.002 ТЗ	Арк.
						2
Зм.	Арк.	№ докум.	Підпис	Дата		

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

- Користувачі повинні мати можливість створювати облікові записи та безпечно входити в програму.
- Програма має підтримувати різні ролі користувачів із різними рівнями доступу та дозволами.
- Користувачі повинні мати можливість створювати, редагувати та керувати проектами, зокрема визначати деталі проекту, призначати членів команди та встановлювати етапи.
- Додаток має дозволяти користувачам створювати, призначати та відстежувати завдання, встановлювати терміни та контролювати хід виконання завдань.

5.2. Вимоги до програмного забезпечення

- Будь-яка ОС: ОС Windows, Mac чи Linux.
- Будь-який браузер
- Можливість виходу до мережі Internet

5.3. Вимоги до апаратної частини

- Процесор не менше ніж Intel® Core (TM) i3-2100T.
- Можливість виходу до мережі Internet
- Оперативна пам'ять від 4 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.12.2022-15.12.2022
Вивчення та аналіз завдання	15.12.2022-15.03.2023
Розробка архітектури та загальної структури системи	15.03.2023-25.03.2023
Розробка структур окремих частин системи	25.03.2023-5.04.2023
Програмна реалізація системи	5.04.2023-15.04.2023
Виправлення помилок	15.04.2023-20.05.2023
Оформлення пояснювальної записки	25.04.2023

					ІАЛЦ.467800.002 ТЗ	Арк.
						3
Зм.	Арк.	№ докум.	Підпис	Дата		

**ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ**

на тему: «Веб-застосунок для управління розробкою проєктів»

Київ – 2023

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ.....	3
ВСТУП	4
РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ СПОСОБІВ ФОРМУВАННЯ ВЕБ-СТОРИНОК.....	5
1.1 Визначення.....	5
1.2 Види формування веб-сторінок.....	6
1.2.1 Client-Side Rendering.....	6
1.2.2 Server-Side Rendering	9
1.2.3 Static-Site Generation	13
1.2.4 Incremental Static Regeneration.....	16
1.3 Визначення найкращого підходу для обраного веб-додатку	20
ВИСНОВОК ДО РОЗДІЛУ 1	23
РОЗДІЛ 2. ОГЛЯД ПРИНЦИПІВ ПОБУДОВИ ВЕБ-СЕРВЕРІВ	25
2.1 Визначення.....	25
2.2 Архітектура веб-сервера.....	26
2.3 Серверні мови програмування.....	29
2.4 Розгортання веб-сервера	31
2.5 Інтеграція баз-даних	33
2.6 Кешування та оптимізація продуктивності.....	35
2.7 Масштабованість і висока доступність.....	37
2.8 Нові тенденції та технології у створенні веб-серверів.....	40
ВИСНОВОК ДО РОЗДІЛУ 2	43
РОЗДІЛ 3. ОГЛЯД РОЗРОБЛЕНОЇ СИСТЕМИ.....	45

					ІАЛЦ.467800.003 ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Веб-застосунок для управління розробкою проектів Пояснювальна записка	Літ.	Аркуш	Аркушів
Розробив	Руденко С.О.						1	80
Перевірив	Русінов В.В.					КПІ ім. Ігоря Сікорського, ФІОТ, ІО-93		
Реценз.								
Н. Контр.	Виноградов Ю.М.							
Затвердив								

3.1 Визначення.....	45
3.2 Обґрунтування обраних засобів розробки - React.....	46
3.3 Обґрунтування обраних засобів розробки - Express	48
3.4 Розробка серверної частини.....	49
3.4.1 Авторизація та реєстрація	49
3.4.2 Методи для керування проектами.....	55
3.4.3 Методи для керування задачами	58
3.4.4 Методи для керування коментарями	63
3.5 Розробка клієнтської частини	66
3.5.1 Огляд	66
3.5.2 Реалізація сторінки з авторизацією.....	68
3.5.3 Огляд бібліотеки MUI	70
3.5.4 Використання користувацьких хуків.....	71
ВИСНОВОК ДО РОЗДІЛУ 3	75
ВИСНОВКИ.....	77
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	79

ПЕРЕЛІК СКОРОЧЕНЬ

SSR	(Server-Side Rendering) Рендеринг на стороні серверу
CSR	(Client-Side Rendering) Рендеринг на стороні клієнту
HTML	(HyperText Markup Language) Стандартизована мова розмітки документів
CSS	(Cascading Style Sheets) Спеціальна мова стилю сторінок
JS	(JavaScript) Мова програмування
JSON	(JavaScript Object Notation) Текстовий формат обміну даними
MUI	(Material UI) Бібліотек React компонентів

					ІАЛЦ.467800.003 ПЗ	Арк.
						3
Зм.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

У сучасному швидкоплинному та взаємопов'язаному світі ефективне управління проектами відіграє ключову роль в успіху організацій у різних галузях. Оскільки компанії здійснюють дедалі складніші проекти з різноманітними командами, зростає потреба в раціоналізованих та ефективних рішеннях для управління розробкою проектів. Це призвело до появи веб-додатків, спеціально розроблених для підтримки процесів управління проектами та сприяння співпраці між членами команди.

Ця бакалаврська робота зосереджена на проектуванні та розробці веб-додатку для управління розробкою проектів. Мета полягає в тому, щоб створити комплексну та зручну платформу, яка забезпечує ефективне планування проектів, керування завданнями, співпрацю та розподіл ресурсів. Веб-додаток дозволить командам проектів ефективно керувати своїми проектами, покращити комунікацію та, зрештою, досягти успішних результатів проекту.

Розробка цієї веб-програми передбачає використання сучасних веб-технологій, фреймворків і найкращих практик для створення інтуїтивно зрозумілої та надійної платформи. Робота включатиме визначення вимог, розробку інтерфейсу користувача, впровадження основної функціональності та ретельне тестування програми для забезпечення якості.

Очікується, що завдяки розробці веб-програми, спеціально розробленої для задоволення потреб проектних груп, організації отримають підвищену продуктивність, покращать співпрацю та підвищать показники успіху проектів. Крім того, ця робота має на меті сприяти зростанню обсягу знань у сфері управління проектами та розробки веб-додатків.

					ІАЛЦ.467800.003 ПЗ	Арк.
						4
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ СПОСОБІВ ФОРМУВАННЯ ВЕБ-СТОРІНОК

1.1 Визначення

Веб-простір став невід’ємною частиною життя кожної сучасної людини. Перегляд прогнозу погоди, новин; комунікація з близькими людьми, клієнтами, розвиток бізнесу – світ побудований навколо веб-сторінок. При створенні додатків розробники повинні враховувати дуже багато чинників – предметну область, наявні технічні ресурси (сервера), майбутню підтримку проектів і т.д. – і основним чинником, що містить в собі рішення більшості поставлених проблем це те, як користувач отримує HTML сторінку. Незважаючи на те, що таке явище як Інтернет існує вже доволі великий проміжок часу, основним підходом було те, що кожен раз, коли користувач робить якусь дію – на сервер відправлявся запит на формування абсолютно нової HTML сторінки, яка відправлялася назад користувачу. Користувачеві раніше доводилося чекати кілька секунд, поки сервер отримує запит, збирає дані, створює HTML і повертає відповідь. Оскільки це було повносторінковим завантаженням, браузер повинен був очікувати всі ресурси, такі як JavaScript, CSS і інші файли, щоб знову їх завантажити (якщо вони не кешуються належним чином). Це становило велику незручність для користувача [1]. Але час іде, технології не стоять на місці і були створені принципово нові концепції, що дозволяють робити ухил на певні конкретні проблеми.

Далі йде перелік основних термінів, що будуть використовуватись при дослідженні питань щодо генерації веб-сторінок:

Вебзастосунок – це застосунок, що складається з двох частин – клієнта (в нашому випадку в ролі клієнта виступає браузер) і в сервера. Вебзастосунок «спілкується» з юзером через інтерфейс веб-сторінки, що відкриває можливості виконувати різноманітні задачі. Серед них - обробка, збереження і

					ІАЛЦ.467800.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		5

відображення даних, взаємодія з БД, автоматизація абсолютно різних бізнес процесів, спільна робота і комунікація.

Рендерінг - це процес, що використовується в веб-розробці, і перетворює код веб-сайту на інтерактивні сторінки, які користувачі бачать під час відвідування веб-сайту. Цей термін зазвичай відноситься до використання HTML, CSS та JavaScript кодів. Процес завершується за допомогою механізму візуалізації - програмного забезпечення, що використовується веб-браузером для відображення веб-сторінки. Завдяки тісному зв'язку з веб-браузерами, механізми візуалізації часто називають механізмами браузера.

1.2 Види формування веб-сторінок

1.2.1 Client-Side Rendering

Рендеринг на клієнтській стороні є відносно новим підходом до створення веб-додатків. Він почав набувати популярності, коли бібліотеки JavaScript почали інтегрувати його (безпосередньо Angular і React.js), які є одними з найкращих прикладів бібліотек, які використовуються в цьому типі рендерингу. Він працює наступним чином – він виконує JavaScript код веб-додатку у вашому браузері, а не на сервері. Сервер в свою чергу надсилає відповідь у вигляді HTML документу, що містить JS код, замість отримання всього документа [2]. Не зважаючи на те, що під час першого завантаження процес відбувається повільно, наступні завантаження сторінок будуть відносно швидшими, адже вони безпосередньо не залежать від іншого документа для кожного роуту. Все, що знаходиться між шарами логіки клієнту до отримання даних від серверу - клієнтські сайти роблять усе в не залежності один від одного (рис. 1.1). Веб-сторінка яку бачить користувач є доступною відразу як результат виконання коду, оскільки кожна сторінка, яку відвідує користувач, і її відповідна URL-адреса створюються динамічно.

					ІАЛЦ.467800.003 ПЗ	Арк.
						6
Зм.	Арк.	№ докум.	Підпис	Дата		

Далі слід розглянути переваги та недоліки такого підходу. З переваг треба в першу чергу виділити те, що CSR дозволяє використовувати динамічні та інтерактивні інтерфейси користувача.

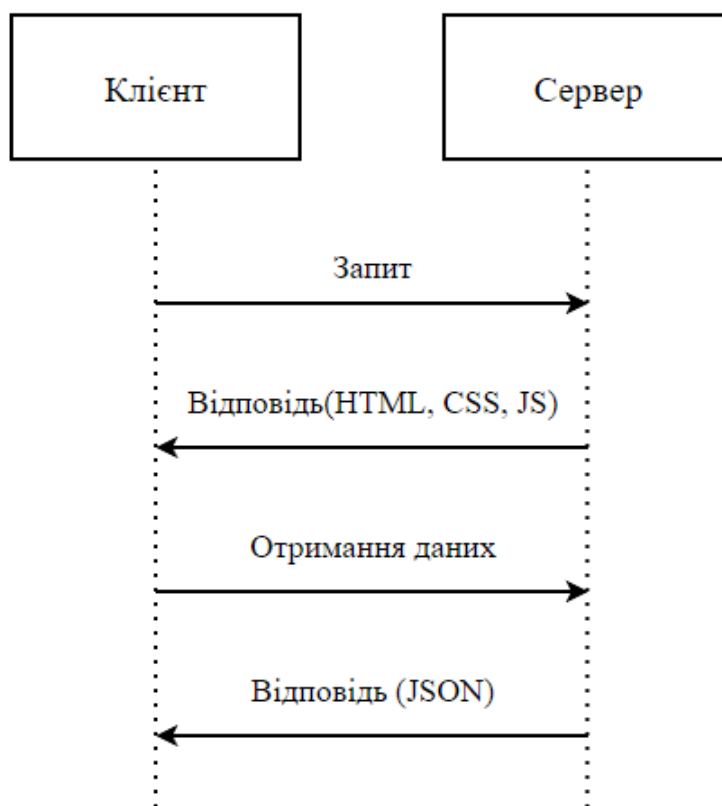


Рисунок 1.1 – Принцип роботи Client-Side Rendering

Процес CSR виглядає наступним чином:

1. Користувач вводить URL-адресу, яку хоче відвідати, в адресному рядку. Запит даних надсилається на сервер за вказаною URL-адресою.
2. Після першого запиту клієнта на сайт сервер доставляє статичні файли (CSS і HTML) у браузер клієнта.
3. Браузер клієнта спочатку завантажить вміст HTML, а потім JavaScript. Ці HTML-файли підключають JavaScript, починаючи процес завантаження, відображаючи користувачеві символи завантаження, визначені розробником. На цьому етапі веб-сайт ще не видимий для користувача.
4. Після завантаження JavaScript вміст динамічно генерується в браузері клієнта.

5. Вміст стає видимим, коли клієнт переміщується веб-сайтом і взаємодіє з ним.

Він забезпечує плавні переходи, анімацію та оновлення в реальному часі без необхідності повного перезавантаження сторінки. Це може призвести до більш коректного і приємного досвіду користувача. Також, оскільки браузер клієнта обробляє візуалізацію, сервер відповідає за надсилання даних у стандартизованому форматі (зазвичай JSON) [3]. Це зменшує обчислювальне навантаження на сервер, оскільки йому не потрібно генерувати HTML для кожного запиту. У результаті сервера можуть обслуговувати більше одночасних користувачів. Ще один момент, вартий уваги - CSR забезпечує ефективне кешування статичних ресурсів, таких як JavaScript, CSS і медіафайлів, на стороні клієнта. Після початкового завантаження цих ресурсів веб-переглядач може кешувати їх локально. Наступні запити можуть обслуговуватися з кешу, зменшуючи потребу в повторних запитах до сервера та покращуючи продуктивність.

Хоча рендеринг на стороні клієнта пропонує кілька переваг, вона також має деякі недоліки, які слід враховувати. CSR значною мірою покладається на JavaScript для відтворення інтерфейсу користувача, що означає, що користувачі, які вимкнули JavaScript або використовують браузери, які його не підтримують, можуть взагалі не мати доступу до програми. Це може обмежити доступність і охоплення програми, проте в наразі проблема може бути не настільки актуальною, адже всі сучасні веб-додатки використовують JavaScript, і досвід користування мережею за відсутності підтримки JS не є можливим. Також, логіка відтворення та керування інтерфейсом користувача знаходиться на стороні клієнта, що призводить до збільшення складності з точки зору архітектури програми та управління її станом. Це може призвести до необхідності обслуговування додаткового коду, потенційних проблем з продуктивністю та проблем з налагодженням.

					ІАЛЦ.467800.003 ПЗ	Арк.
						8
Зм.	Арк.	№ докум.	Підпис	Дата		

Отже, важливо ретельно оцінити конкретні вимоги та компроміси проекту, перш ніж вибрати між CSR та іншими підходами до візуалізації, такими як рендеринг на стороні сервера або гібридний рендеринг, щоб забезпечити найкращу відповідність потребам програми

1.2.2 Server-Side Rendering

Візуалізація на стороні сервера (SSR) — це підхід у веб-розробці, коли сервер створює повністю згенеровані сторінки HTML і надсилає їх клієнту у відповідь на запит. У SSR сервер обробляє запит, отримує дані та динамічно генерує вміст HTML, який представляє запитувану сторінку.

Традиційно веб-програми покладалися на рендеринг на стороні клієнта (CSR), коли початкова HTML-сторінка надсилається із сервера як скелет, а потім код JavaScript на боці клієнта бере на себе функцію отримання даних і відтворення вмісту в браузері. Такий підхід може призвести до сповільнення початкового часу завантаження сторінки та може призвести до неоптимальної продуктивності для індексування пошуковою системою та спільного використання в соціальних мережах [4].

При рендерингу на стороні сервера сервер відповідає за створення повної HTML-розмітки сторінки з початковими даними (рис 1.2). Це означає, що коли клієнт отримує HTML-код із сервера, він уже містить повністю відтворений вміст. Браузер клієнта може відобразити сторінку, не чекаючи, поки додатковий JavaScript завантажить і відтворить вміст.

Однією з основних переваг SSR є швидке початкове завантаження сторінки. За допомогою SSR сервер створює повністю відрендерену HTML-сторінку, яку можна надіслати клієнту. Це означає, що користувачі можуть бачити вміст швидше, оскільки їм не потрібно чекати, поки файли JavaScript завантажаться та запускаються перед відтворенням сторінки.

Попередньо відрендерений HTML відразу стає видимим для користувача, забезпечуючи швидшу та безперебійну початкову роботу [5].

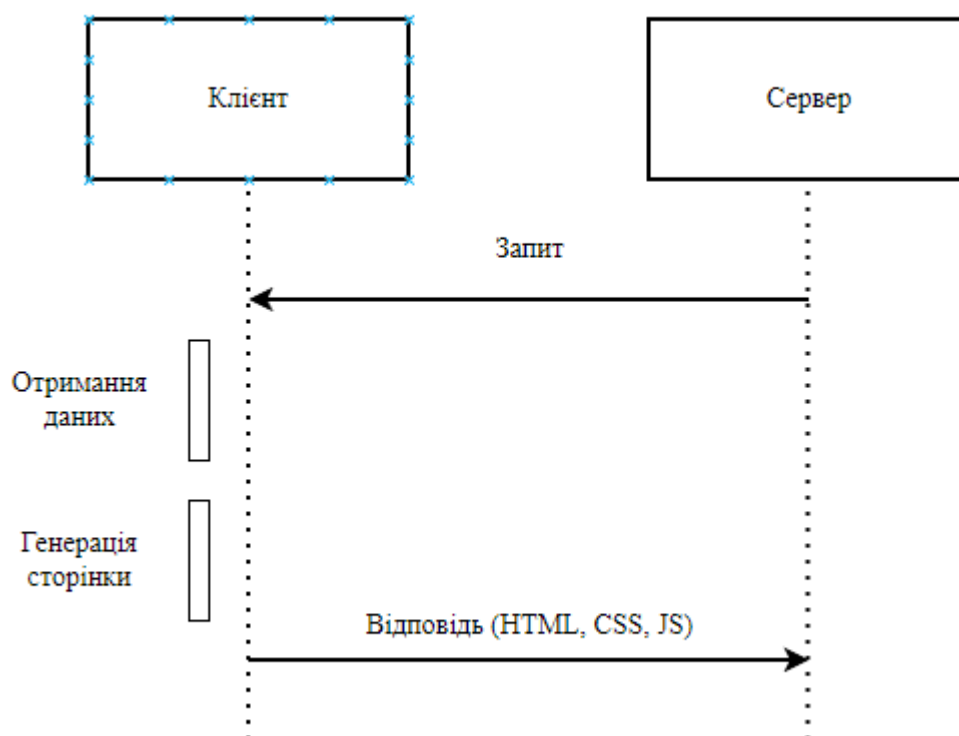


Рисунок 1.2 – Принцип роботи Server-Side Rendering

Загальні кроки SSR такі:

1. Браузер надсилає запит щодо певної вебсторінки або маршруту вебсайту.
2. Сервер отримує запит і починає його обробку. Це може включати отримання даних із баз даних або зовнішніх API, виконання обчислень на стороні сервера або виконання інших необхідних операцій.
3. За допомогою SSR сервер використовує отримані дані та будь-яку іншу необхідну інформацію для створення повної структури HTML запитованої сторінки. Це включає відтворення вмісту, застосування стилів і додавання будь-яких динамічних елементів.
4. Після того, як завершиться рендеринг, сервер надсилає повністю згенеровану HTML-сторінку назад браузеру як відповідь на початковий запит.

5. Браузер клієнта отримує відповідь у форматі HTML і відображає попередньо відтворений вміст для користувача. Оскільки HTML уже готовий, користувач може негайно побачити повністю відтворену сторінку, включаючи її вміст і макет.

SSR у веб-розробці має певну кількість переваг над іншими підходами. Нижче на рис 1.3 приведені графіки з сайту DebugBear, який відображає моніторинг швидкості роботи веб-додатку

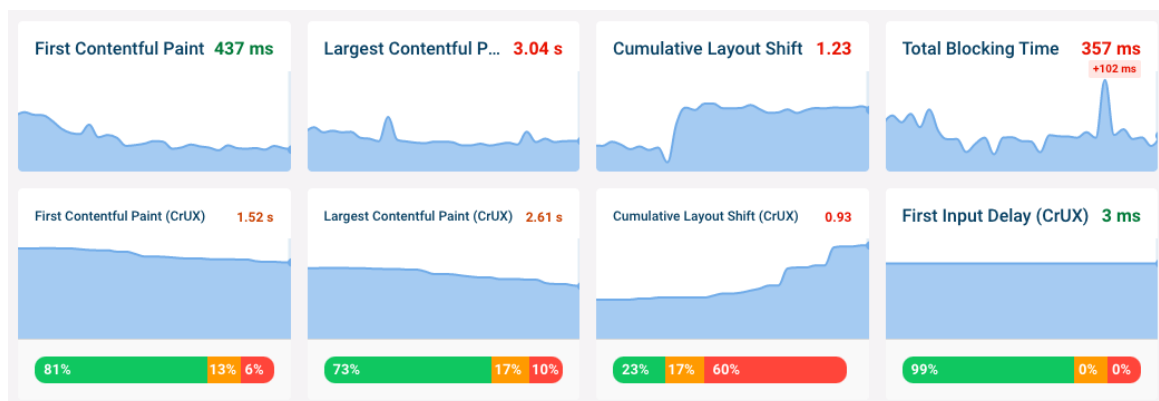


Рисунок 1.3 – Графіки зі швидкістю роботи додатку, що використовує SSR [6].

Ще однією перевагою SSR є покращена пошукова оптимізація (SEO). Пошукові системи можуть легко сканувати та індексувати повністю відтворені сторінки HTML. Це покращує видимість додатку в результатах пошуку, оскільки вміст легко доступний для індексації. SSR гарантує, що пошукові системи можуть точно читати й аналізувати вміст, що може позитивно вплинути на роботу веб-сайту з SEO.

SSR також покращує взаємодію з користувачем, дозволяючи користувачам переглядати вміст і взаємодіяти з веб-сайтом без затримок. Попередньо відтворений HTML гарантує, що початкова сторінка з'явиться швидко та буде повністю придатною для використання. Це може сприяти покращенню взаємодії з користувачем, особливо для відвідувачів із повільнішим підключенням до Інтернету або пристроями з обмеженою обчислювальною потужністю.

З точки зору розробки та обслуговування, SSR може спростити процес. Відокремивши логіку презентації від коду JavaScript на стороні клієнта, розробники можуть зосередитися на створенні надійних компонентів на стороні сервера, покладаючись на сервер для обробки візуалізації. Це розділення може призвести до чистішого коду та полегшення обслуговування в довгостроковій перспективі.

В свою чергу, рендеринг на стороні сервера має деякі недоліки, які слід враховувати. Одним із головних недоліків SSR є потенційне збільшення навантаження на сервер. Створення повністю відтворених HTML-сторінок на сервері потребує більше обчислювальних ресурсів порівняно з обслуговуванням статичних файлів. Сервер повинен обробляти кожен запит, отримувати дані та динамічно генерувати вміст HTML, що може створювати навантаження на сервер, особливо в періоди високого трафіку. Іншим недоліком SSR є те, що подальша взаємодія з веб-сайтом може бути повільнішою порівняно з рендерингом на стороні клієнта. Оскільки SSR зазвичай передбачає повне перезавантаження сторінки для кожної дії користувача, веб-сайт може здаватися менш чуйним та інтерактивним. Це може бути помітно під час навігації між сторінками або виконання динамічних дій, які вимагають оновлення без повного перезавантаження.

Реалізація SSR може вимагати додаткових потужностей і ресурсів на стороні сервера. Сервер повинен обробляти візуалізацію та генерацію HTML-сторінок, для чого може знадобитися більш надійна інфраструктура або серверні можливості порівняно з обслуговуванням статичних файлів. Це може призвести до збільшення витрат і ускладнень експлуатації.

Крім того, SSR може не підходити для всіх типів програм або випадків використання. Веб-сайти або програми, які значною мірою покладаються на візуалізацію на стороні клієнта та динамічне оновлення вмісту, можуть не отримати такої користі від SSR. Наприклад, додатки з високоінтерактивними користувацькими інтерфейсами або функціями співпраці в реальному часі

					ІАЛЦ.467800.003 ПЗ	Арк.
						12
Зм.	Арк.	№ докум.	Підпис	Дата		

можуть краще обслуговуватися клієнтськими платформами візуалізації або гібридними підходами до візуалізації.

Загалом, рендеринг на стороні сервера позитивно впливає на час завантаження сторінки, SEO, підтримку соціальних мереж і доступність. З іншого боку, візуалізація на стороні клієнта може бути дешевшою, легшою для створення та обслуговування та кращою для затримки першого введення (FID – First Input Delay).

1.2.3 Static Site Generation

SSG — це інструмент, який генерує повний статичний HTML-сайт на основі необроблених даних і набору шаблонів (рис 1.4). По суті, генератор статичного сайту автоматизує завдання кодування окремих HTML-сторінок і завчасно готує ці сторінки до показу користувачам.

Оскільки ці сторінки HTML є попередньо створеними, вони можуть дуже швидко завантажуватися у браузерях користувачів.

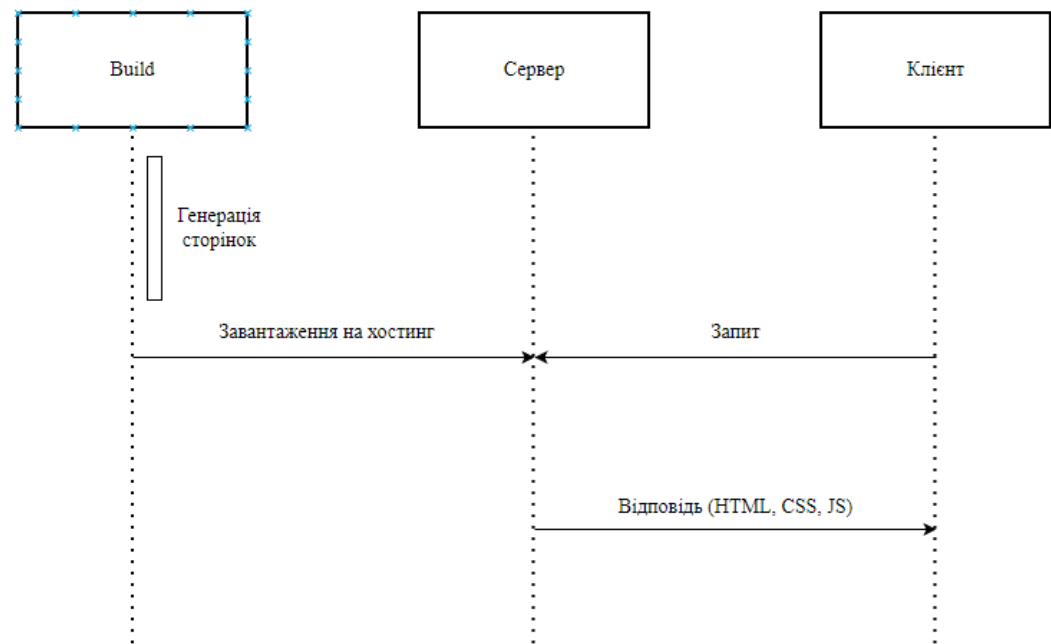


Рисунок 1.4 – Принцип роботи Static Site Generation

Генератори статичних сайтів пропонують кілька переваг порівняно з динамічними веб-сайтами чи іншими підходами. Ось деякі з ключових переваг:

1. Статичні сайти працюють швидко, оскільки вони попередньо відображаються й не вимагають обробки на стороні сервера. Це призводить до швидшого часу завантаження, покращення взаємодії з користувачем і кращої пошукової оптимізації (SEO). Оскільки вміст подається у вигляді статичних файлів, можна легко реалізувати стратегії кешування, що ще більше підвищує продуктивність.
2. Статичні сайти за своєю суттю безпечніші, ніж динамічні. За відсутності виконання коду на стороні сервера або взаємодії з базою даних є менше можливостей для вразливостей і спроб злому. Це спрощує налаштування безпеки та зменшує поверхню атаки.
3. Генератори статичних сайтів використовують підхід на основі файлів, що полегшує керування та впорядкування вмісту. Шаблони та повторно використововувані компоненти можна використовувати для підтримки узгодженості на сайті. Простота також поширюється на робочі процеси розробки та розгортання, що призводить до пришвидшення циклів розробки.
4. Статичні сайти часто вимагають менше ресурсів сервера, що зменшує витрати на хостинг. Крім того, немає потреби в серверній інфраструктурі або базах даних, що ще більше знижує операційні витрати.
5. Генератори статичних сайтів пропонують широкий спектр варіантів налаштування, дозволяючи розробникам вибирати інструменти, фреймворки та бібліотеки, які відповідають їхнім потребам. Вони можуть інтегруватися з API, використовувати фреймворки JavaScript для інтерактивності та використовувати плагіни та теми, надані екосистемою генератора.

Водночас, SSG мають багато недоліків, які слід враховувати. В першу чергу, на статичних сайтах немає оновлень у реальному часі, і вони не можуть динамічно змінюватися на основі даних у реальному часі чи взаємодії

					ІАЛЦ.467800.003 ПЗ	Арк.
						14
Зм.	Арк.	№ докум.	Підпис	Дата		

користувача. Керування вмістом у генераторах статичних сайтів може бути складним, оскільки для редагування або створення вмісту у вихідних файлах або шаблонах часто потрібні глибокі технічні знання, і реалізація цієї логіки є доволі проблемною.

Інтерактивність як така обмежена на статичних сайтах, які покладаються переважно на статичні файли HTML, CSS і JavaScript. Хоча JS може додати деяку інтерактивність, він має обмеження порівняно з фреймворками, що орієнтуються на підходи генерування HTML документів на стороні серверу.

Створення статичних сайтів може зайняти багато часу, особливо для великих сайтів або проектів зі складною конфігурацією. Налаштування процесу збірки та обробки залежностей може додати складності.

Масштабованість може бути проблемою для статичних сайтів із частим оновленням вмісту. Кожна зміна вимагає перебудови та повторного розміщення всього сайту, що може спричинити затримки. Якщо сайт містить близько 100 000 сторінок, то генерація може зайняти декілька годин

1.2.4 Incremental Static Regeneration

Інкрементна статична регенерація (ISR) — це функція, яку пропонують деякі генератори статичних сайтів і фреймворки, яка дозволяє динамічно оновлювати вміст на статично згенерованих веб-сайтах. Він поєднує в собі переваги статичних сайтів, такі як висока продуктивність і безпека, з можливістю включати дані в реальному часі або дані, які часто змінюються [7].

Однак замість того, щоб вимагати повної перебудови сайту для оновлення вмісту, ISR дозволяє повторно генерувати певні сторінки чи розділи на вимогу або через заздалегідь визначені проміжки часу (рис 1.5).

Принцип його роботи наступний:

1. Під час початкового процесу створення статичний генератор сайту генерує файли HTML для сторінок веб-сайту на основі доступних даних на той момент. Це створює повністю статичну версію сайту.

					ІАЛЦ.467800.003 ПЗ	Арк.
						15
Зм.	Арк.	№ докум.	Підпис	Дата		

2. Коли надсилається запит на динамічно згенеровану сторінку чи розділ, генератор статичного сайту запускає поступове оновлення. Він отримує найновіші дані або виконує обчислення, необхідні для створення оновленого вмісту.
3. Замість того, щоб відновлювати весь сайт, відновлюється лише уражена сторінка чи розділ. Це мінімізує час і ресурси, необхідні для оновлення.
4. Потім оновлений вміст відображається та кешується як новий статичний файл. Подальші запити для цієї сторінки чи розділу можна обслуговувати безпосередньо з кешу, усуваючи потребу в подальшій регенерації, доки дані не зміняться знову.

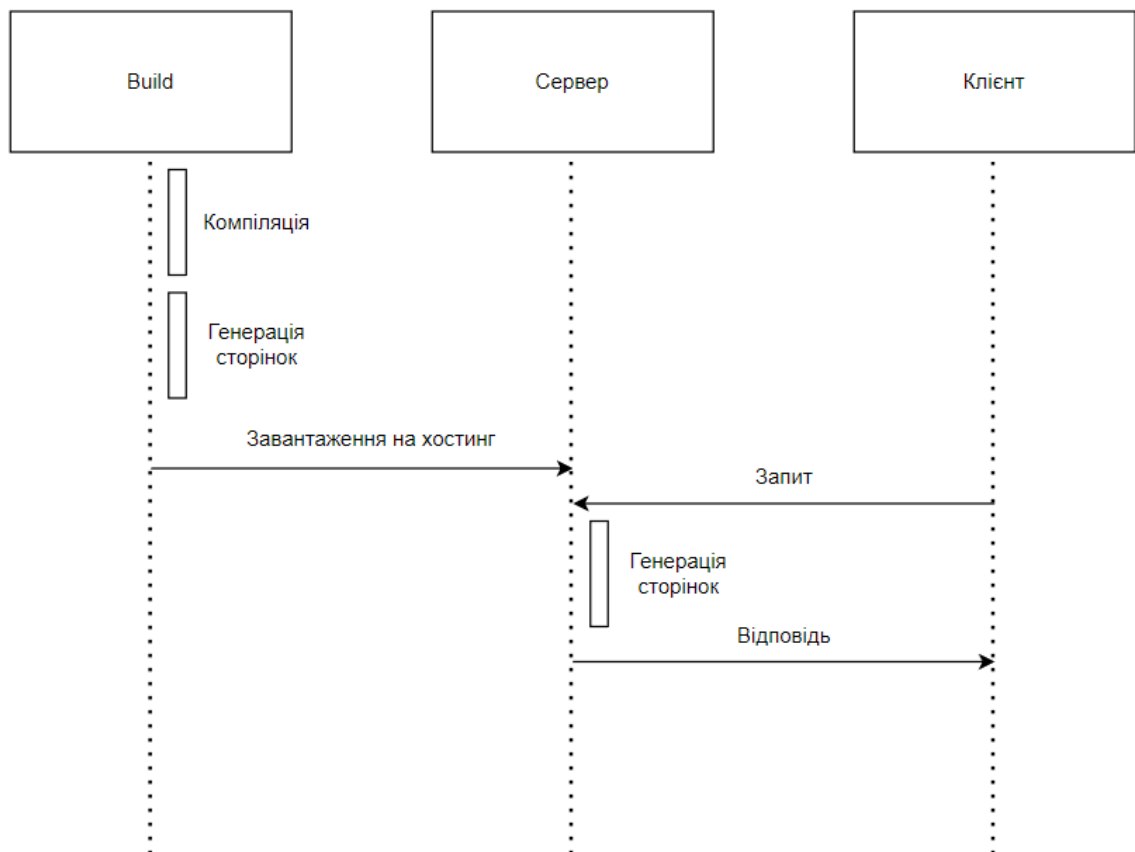


Рисунок 1.5 – Принцип роботи Incremental Static Regeneration

За допомогою ISR вибрані сторінки або розділи веб-сайту генеруються статично під час процесу створення, так само, як у традиційному статичному створенні сайту.

Однією з істотних переваг ISR є покращена продуктивність. Генерація статичного сайту дозволяє швидше завантажувати сторінки завдяки попередній візуалізації сторінок як статичних файлів. За допомогою ISR наступні оновлення можна робити за вимогою, зменшуючи потребу у перебудові всього сайту. Це призводить до швидшого оновлення та загального покращення продуктивності.

Ще однією ключовою перевагою є можливість включати динамічні оновлення вмісту. Хоча на статичних сайтах традиційно відсутні динамічні елементи, ISR дозволяє динамічно оновлювати певні сторінки, коли стає доступним новий вміст. Це означає, що нові дані, такі як публікації в блозі, можна додавати без перебудови всього сайту, забезпечуючи оновлення в режимі реального часу або майже в режимі реального часу, не жертвуючи перевагами створення статичного сайту.

Масштабованість і економічність також є перевагами ISR. Статичні сайти можна легко обслуговувати мережами доставки контенту (CDN), а ISR зменшує потребу в обробці на стороні сервера, обмежуючи динамічні оновлення окремими сторінками. Це призводить до зменшення навантаження на сервер, покращеної масштабованості та нижчих витрат на інфраструктуру порівняно з повністю динамічними веб-сайтами.

Кешування та автономна підтримка є додатковими перевагами ISR. Статичні сайти можна ефективно кешувати та швидко обслуговувати з периферійних серверів. Використовуючи ISR, певні сторінки можна кешувати та обслуговувати, навіть якщо серверна частина сайту тимчасово недоступна. Це покращує стійкість, офлайн-підтримку та загальну взаємодію з користувачем.

ISR забезпечує гнучкість джерел даних, дозволяючи інтеграцію з базами даних, API або іншими зовнішніми системами для отримання динамічного вмісту. Це означає, що розробники можуть поєднувати статичні дані з динамічними оновленнями, гарантуючи, що сайт залишається в актуальному

					ІАЛЦ.467800.003 ПЗ	Арк.
						17
Зм.	Арк.	№ докум.	Підпис	Дата		

стані, водночас користуючись перевагами продуктивності створення статичного сайту.

Нарешті, ISR надає переваги SEO. Статичні сайти відомі своїми перевагами для SEO завдяки швидкому завантаженню та чіткій структурі HTML. ISR дозволяє додавати динамічний вміст до статичних сторінок, покращуючи SEO, забезпечуючи пошуковим системам можливість виявлення та індексування найновішого вмісту на сайті.

Хоча ISR пропонує численні переваги, важливо враховувати можливі недоліки під час впровадження цієї техніки. Розуміння недоліків може допомогти розробникам прийняти зважені рішення щодо його придатності для їхніх проектів. Ось кілька ключових:

- Одним з недоліків ISR є підвищена складність. Впровадження ISR потребує додаткової конфігурації та налаштування порівняно з традиційним статичним створенням сайту. Управління процесом поступової регенерації, обробка динамічних оновлень і забезпечення узгодженого кешування та стратегій отримання даних можуть ускладнити процес розробки.
- Інтерактивність у реальному часі обмежена ISR. Хоча ISR дозволяє динамічні оновлення, частота оновлень залежить від наявності нового вмісту або певних тригерів. Програми, які потребують даних у реальному часі або миттєвої взаємодії з користувачем, можуть не підходити для ISR.
- З ISR можуть виникнути проблеми з кешуванням. Управління стратегіями кешування стає складнішим, щоб забезпечити надання користувачам правильних версій сторінок. Неправильні конфігурації або невідповідності в кешуванні можуть призвести до доставки застарілого або неправильного вмісту, що вплине на точність і надійність сайту.

					ІАЛЦ.467800.003 ПЗ	Арк.
						18
Зм.	Арк.	№ докум.	Підпис	Дата		

- Нарешті, ISR вводить накладні витрати на розробку та обслуговування. Ця техніка вносить додаткову складність і вимагає від розробників ознайомлення з процесом поступової регенерації, механізмами отримання даних, стратегіями кешування та потенційними проблемами, які можуть виникнути. Поточне технічне обслуговування та усунення несправностей динамічних оновлень може бути більш складним порівняно з повністю статичними сайтами.

Підсумовуючи, інкрементна статична регенерація — це потужна техніка, яка поєднує в собі переваги створення статичного сайту з можливістю включати динамічні оновлення. Цей підхід пропонує кілька переваг, включаючи покращену продуктивність, динамічне оновлення вмісту, масштабованість, економічну ефективність, переваги кешування, гнучкість джерел даних і вдосконалення SEO. Вибірково оновлюючи конкретні сторінки, зберігаючи переваги статичної архітектури сайту, ISR забезпечує швидший час завантаження сторінок, оновлення в режимі реального часу або майже в режимі реального часу, зменшує навантаження на сервер, покращує взаємодію з користувачем і покращує видимість пошукової системи.

Однак важливо враховувати можливі недоліки ISR. До них відносяться підвищена складність реалізації, обмежена інтерактивність у реальному часі, можливість застарілого вмісту під час затримок оновлення, потенційний вплив на час створення для конкретних сторінок, проблеми з кешуванням, використання ресурсів сервера та збільшення витрат на розробку та обслуговування.

Розглядаючи ISR для конкретного проекту, дуже важливо оцінити вимоги, цілі та компроміси. Слід ретельно розглянути потреби проекту в інтерактивності в реальному часі, свіжості вмісту, толерантності до складності та впливі на робочі процеси розробки та поточне обслуговування.

Загалом ISR пропонує ефективне рішення для балансу між перевагами створення статичного сайту та необхідністю динамічного оновлення. Це цінна

					ІАЛЦ.467800.003 ПЗ	Арк.
						19
Зм.	Арк.	№ докум.	Підпис	Дата		

техніка в сучасній веб-розробці, яка може значно покращити продуктивність, взаємодію з користувачем, масштабованість і пошукову оптимізацію веб-сайтів. Використовуючи потужність ISR, розробники можуть створювати швидкі, гнучкі та ефективні веб-додатки, які забезпечують своєчасне оновлення та захоплюючий досвід для користувачів [8].

1.3 Визначення найкращого підходу для обраного веб-додатку

Для веб-додатку, який відповідає за керування проектами, поєднання візуалізації на стороні сервера і візуалізації на стороні клієнта може бути найефективнішим методом формування веб-сторінок.

SSR корисна для початкового завантаження сторінки та оптимізації пошукової системи. Він включає генерацію HTML на сервері та доставку повністю відтворених сторінок клієнту. Коли користувач отримує доступ до веб-додатку, він отримує повний документ HTML із сервера, що скорочує час, необхідний для завантаження початкової сторінки. SSR ідеально підходить для додатків управління проектами, оскільки забезпечує швидшу візуалізацію та кращу оптимізацію пошукових систем (SEO), завдяки чому інформація про проект стає легкодоступною для користувачів і легко виявляється пошуковими системами [9].

На таблиці 1. можна побачити порівняльну таблицю SSR та CSR, де порівнюються різні часові показники, які є вирішальними при розробці та користуванні веб-додатком.

CSR має вирішальне значення для інтерактивності та динамічних оновлень у додатку для керування проектами.

Цей гібридний підхід забезпечує баланс між продуктивністю, SEO та інтерактивністю. Це забезпечує швидке завантаження програми для керування проектами, її легко знайти пошуковими системами, а також забезпечує багатий і чутливий досвід користувача.

					ІАЛЦ.467800.003 ПЗ	Арк.
						20
Зм.	Арк.	№ докум.	Підпис	Дата		

CSR передбачає відтворення вмісту в браузері клієнта за допомогою JavaScript. Це забезпечує більш інтерактивний і оперативний досвід користувача, забезпечуючи оновлення в реальному часі, динамічну фільтрацію вмісту та розширену взаємодію з користувачем. Завдяки CSR користувачі можуть переглядати та керувати даними проекту без необхідності перезавантажувати всю сторінку, що забезпечує більш плавну та привабливу роботу.

Таблиця 1.1 - Порівняльна таблиця CSR та SSR

	SSR	CSR
Initial Page Load Time	2.5 seconds	3.5 seconds
Time to Interactive	4 seconds	5 seconds
Subsequence Page Loads	1.8 seconds	1.5 seconds
Bandwidth Usage	850 KB	950 KB
Search Engine Optimization	Good SEO performance with pre-rendered HTML content	May require additional steps
Development Complexity	Moderate complexity, relying on server-side rendering techniques	Higher complexity, managing client-side rendering logic, data fetching, and updates on the client

Поєднуючи SSR і CSR, веб-додаток може використовувати сильні сторони обох підходів. Початкове завантаження сторінки виграє від SSR, забезпечуючи швидке та повне відтворення, тоді як подальші взаємодії та оновлення обробляються CSR, що забезпечує динамічну та адаптивну функціональність.

Щоб реалізувати цю комбінацію, програма керування проектами може використовувати SSR для відтворення інформаційної панелі проекту, списків проектів і окремих деталей проекту. Ці сторінки попередньо відображаються на сервері та доставляються клієнту як повністю сформовані документи HTML. Після завантаження клієнтський JavaScript починає працювати, дозволяючи користувачам взаємодіяти з функціями керування проектами, такими як створення та редагування завдань, співпраця в режимі реального часу та оновлення статусу проекту.

Однак важливо зазначити, що вибір методів візуалізації повинен керуватися конкретними вимогами та міркуваннями програми керування проектами. Такі фактори, як складність програми, потреба в оновленні в режимі реального часу та поведінка цільової аудиторії під час перегляду, слід ретельно оцінити, щоб визначити найбільш підходящий підхід для формування веб-сторінок у контексті програми для керування проектами.

					ІАЛЦ.467800.003 ПЗ	Арк.
						22
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було надано огляд існуючих методів формування веб-сторінок, а саме візуалізації на стороні сервера (SSR), візуалізації на стороні клієнта (CSR), інкрементної статичної регенерації (ISR) і статичної генерації сайту (SSG). Кожен метод пропонує унікальні переваги та недоліки для розробників під час створення веб-додатків.

Візуалізація на стороні сервера (SSR) дозволяє генерувати HTML на сервері та надавати клієнту повністю відрендерені сторінки. Він забезпечує кращий початковий час завантаження, оптимізацію пошукової системи та ефективну деградацію для користувачів із обмеженою підтримкою JavaScript. Однак SSR може потребувати ресурсів на сервері та мати обмеження щодо інтерактивності та оновлень у реальному часі.

Відтворення на стороні клієнта (CSR) покладається на JavaScript для відтворення вмісту в браузері клієнта, забезпечуючи більш інтерактивну та динамічну взаємодію з користувачем. Це переносить обробку на клієнта, зменшуючи навантаження на сервер і забезпечуючи багаті інтерфейси користувача. Однак CSR може призвести до повільнішого початкового завантаження сторінки, потенційних проблем із пошуковою системою пошукових систем і обмеження доступності для користувачів із браузерами, у яких відключено JavaScript.

Інкрементна статична регенерація (ISR) поєднує в собі переваги створення статичного сайту та динамічного оновлення. Це дозволяє оновлювати певні сторінки за вимогою, використовуючи переваги попередньо відрендерених статичних сторінок. ISR покращує продуктивність, пропонує оновлення в реальному або майже реальному часі, а також покращує масштабованість і кешування. Однак реалізація ISR вимагає додаткової складності та ретельного керування кешуванням і отриманням даних.

					ІАЛЦ.467800.003 ПЗ	Арк.
						23
Зм.	Арк.	№ докум.	Підпис	Дата		

Static Site Generation (SSG) попередньо генерує статичні HTML-сторінки під час створення, що забезпечує швидке завантаження, відмінну пошукову оптимізацію та високий рівень безпеки. SSG спрощує процес розгортання та дозволяє розміщення в мережах доставки вмісту (CDN). Однак динамічні оновлення вмісту зазвичай вимагають перебудови всього сайту, що потенційно може призвести до більш тривалого часу створення та проблем з інтерактивністю в реальному часі.

Вибір найбільш прийняттого методу залежить від вимог проекту, таких як цілі продуктивності, потреби в інтерактивності, свіжість вмісту, масштабованість і міркування щодо SEO. Вибір відповідного методу візуалізації має вирішальне значення для досягнення оптимальної взаємодії з користувачем і досягнення цілей проекту.

Майбутні дослідження та прогрес у веб-розробці можуть запровадити нові методи візуалізації або вдосконалити існуючі методи, усунувши їх відповідні обмеження та пропонуючи ще більш гнучкі та продуктивні підходи до формування веб-сторінок. Бути в курсі галузевих тенденцій і постійно оцінювати методи візуалізації буде важливо для розробників, щоб створювати привабливі веб-додатки в цифровому середовищі, що постійно розвивається.

					ІАЛЦ.467800.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		24

РОЗДІЛ 2. ОГЛЯД ПРИНЦИПІВ ПОБУДОВИ ВЕБ-СЕРВЕРІВ

2.1 Визначення

Швидкий розвиток Інтернету та зростання попиту на веб-додатки викликали необхідність глибшого розуміння принципів, що лежать в основі створення ефективних і масштабованих веб-серверів. Веб-сервери утворюють основу інфраструктури Інтернету, відповідальні за обробку запитів клієнтів, обробку даних і своєчасну та надійну доставку веб-вмісту. Щоб побудувати надійні та високопродуктивні веб-сервери, розробники повинні розуміти фундаментальні принципи та методи, задіяні в їх створенні [11].

Метою цього розділу є поглиблений огляд принципів побудови веб-серверів. Він спрямований на вивчення основоположних концепцій, архітектурних міркувань і ключових компонентів, які сприяють розробці та реалізації веб-серверів. Досліджуючи різні методології та передовий досвід, ця теза дасть читачам повне розуміння основних принципів, необхідних для побудови надійних і ефективних систем веб-серверів.

Розділ розпочнеться із заглиблення в основні концепції архітектури веб-сервера, включаючи модель клієнт-сервер, протокол HTTP та цикл запит-відповідь. Він досліджуватиме різні серверні мови програмування, фреймворки та бібліотеки, які зазвичай використовуються у розробці веб-серверів. Крім того, у дисертації обговорюватиметься значення паралельності та масштабованості веб-серверів, а також розглядатимуться різні методи, які використовуються для обробки одночасних з'єднань та оптимізації продуктивності сервера.

Крім того, у розділі досліджуватимуться важливі аспекти, такі як балансування навантаження, механізми кешування, міркування щодо безпеки та методи відмовостійкості, які використовуються для створення надійних і стійких веб-серверів. Він досліджуватиме роль інтеграції бази даних, керування

					ІАЛЦ.467800.003 ПЗ	Арк.
						25
Зм.	Арк.	№ докум.	Підпис	Дата		

сеансами та механізмів автентифікації в покращенні функціональності сервера та взаємодії з користувачем.

Приклади з реального світу та тематичні дослідження будуть використані для ілюстрації застосування обговорюваних принципів у практичних сценаріях. Аналізуючи існуючі технології веб-серверів і вивчаючи їх варіанти дизайну, сильні сторони та обмеження, читачі отримають цінну інформацію про компроміси, пов'язані з розробкою веб-серверів.

Знання та розуміння, отримані в цій дипломній роботі, дозволять розробникам приймати обґрунтовані рішення щодо вибору відповідних технологій і шаблонів проектування для створення ефективних і масштабованих веб-серверів. Дотримуючись принципів, викладених у цій дипломній роботі, розробники матимуть кращу підготовку для вирішення проблем, пов'язаних із розробкою веб-серверів, і сприятимуть створенню надійних і високопродуктивних веб-додатків [12].

2.2 Архітектура веб-сервера

Веб-сервери є основою всесвітньої павутини, відповідальні за доставку веб-контенту користувачам. Розуміння фундаментальної архітектури веб-серверів має важливе значення для створення та підтримки надійних веб-додатків. Зазвичай архітектура складається з кількох взаємопов'язаних компонентів (рис 2.1).

В основі архітектури веб-сервера лежить апаратне забезпечення. Веб-серверам потрібне спеціальне обладнання для обробки вхідних запитів і обробки відповідей. Це включає сервери, пристрої зберігання даних, мережеву інфраструктуру та балансувальники навантаження. Апаратне забезпечення слід ретельно вибирати, щоб відповідати очікуваному трафіку та вимогам веб-додатку до ресурсів.

					ІАЛЦ.467800.003 ПЗ	Арк.
						26
Зм.	Арк.	№ докум.	Підпис	Дата		

Операційні системи відіграють вирішальну роль в архітектурі веб-сервера. Веб-сервери працюють на базі операційної системи (ОС), яка керує системними ресурсами та забезпечує платформу для виконання серверних процесів. Загальні параметри ОС сервера включають дистрибутиви Linux (наприклад, Ubuntu, CentOS) і Windows Server. Вибір операційної системи залежить від таких факторів, як продуктивність, стабільність і сумісність із серверним програмним забезпеченням.

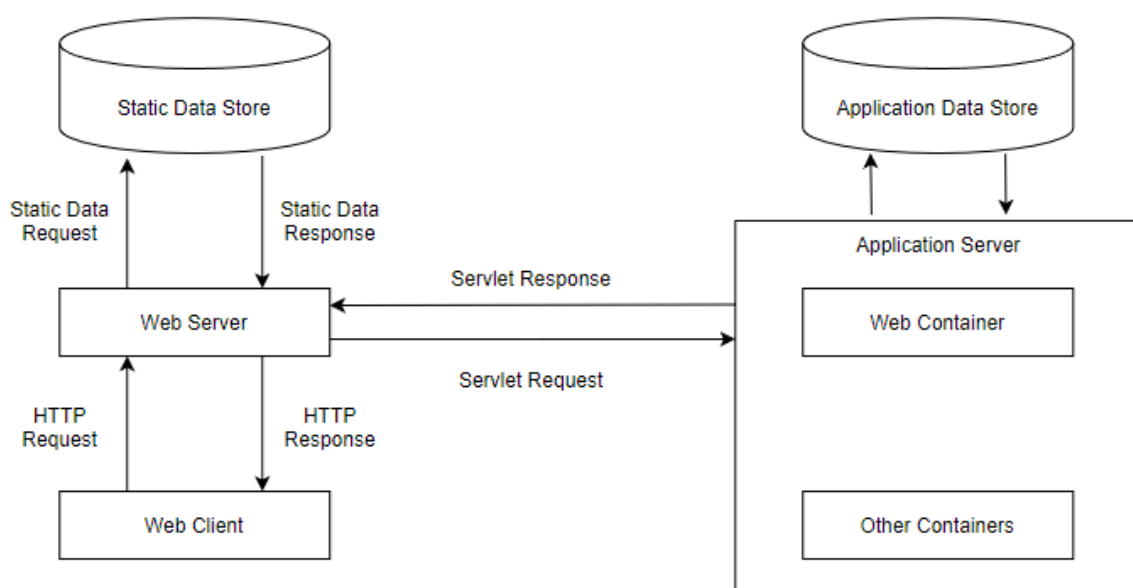


Рисунок 2.1 – Огляд роботи веб-додатку

Програмне забезпечення веб-сервера є серцевиною архітектури веб-сервера. Він приймає запити клієнтів і надає веб-контент. Популярне програмне забезпечення для веб-серверів включає Apache HTTP Server, Nginx і Microsoft Internet Information Services (IIS). Ці сервери відповідають за обробку вхідних запитів, їх маршрутизацію до відповідних програм на стороні сервера та надання відповідних відповідей. Вони забезпечують необхідну інфраструктуру для розміщення веб-сайтів і веб-додатків.

Мережа є ключовим аспектом архітектури веб-сервера. Веб-сервери покладаються на мережеві протоколи для зв'язку з клієнтами та іншими серверами. Програмне забезпечення веб-сервера використовує такі протоколи,

як HTTP (протокол передачі гіпертексту) або HTTPS (HTTP Secure) для обробки запитів і відповідей. Мережеві компоненти, такі як маршрутизатори, брандмауери та балансувальники навантаження, оптимізують мережевий трафік, покращують безпеку та розподіляють вхідні запити між кількома серверами.

Безпека має величезне значення в архітектурі веб-сервера. Веб-сервери потребують надійних заходів безпеки для захисту від потенційних загроз. Це включає впровадження протоколів шифрування (наприклад, SSL/TLS) для безпечної передачі даних, налаштування контролю доступу для обмеження неавторизованого доступу та використання механізмів для запобігання типовим уразливостям, таким як міжсайтовий сценарій (XSS) або атаки SQL Injection. Добре розроблена система безпеки забезпечує цілісність і конфіденційність даних, якими обмінюються клієнти та сервери [13].

Моніторинг і ведення журналів необхідні для безперебійної роботи веб-серверів. Інструменти моніторингу збирають і аналізують показники сервера, відстежують помилки та спостерігають за використанням ресурсів. Це допомагає адміністраторам забезпечити оптимальну продуктивність, виявити вузькі місця продуктивності та усунути проблеми. Інструменти журналювання записують діяльність сервера, включаючи запити, відповіді та помилки, надаючи цінний ресурс для діагностики та вирішення проблем.

Розуміння фундаментальної архітектури веб-серверів, включаючи апаратне забезпечення, операційні системи, програмне забезпечення веб-сервера, сервери додатків, бази даних, мережу, безпеку та компоненти моніторингу/реєстрації, має вирішальне значення для створення надійних і масштабованих веб-додатків. Він дає змогу розробникам і адміністраторам проектувати, налаштовувати та оптимізувати середовища веб-сервера для забезпечення ефективної, безпечної та високопродуктивної роботи в Інтернеті.

					ІАЛЦ.467800.003 ПЗ	Арк.
						28
Зм.	Арк.	№ докум.	Підпис	Дата		

2.3 Серверні мови програмування

Серверні мови програмування та фреймворки відіграють вирішальну роль у створенні динамічних та інтерактивних веб-додатків. Ці технології забезпечують основу для розробки на стороні сервера, дозволяючи розробникам керувати бізнес-логікою, обробляти дані та спілкуватися з базами даних. Ця стаття досліджує ландшафт серверних мов програмування та фреймворків, висвітлюючи їхні особливості, переваги та використання. Розуміння цих технологій має важливе значення для веб-розробників, щоб вибрати найбільш підходящі інструменти для своєї роботи та майбутніх проектів [14].

PHP — це широко поширена серверна мова сценаріїв, спеціально розроблена для веб-розробки. Він пропонує широку екосистему фреймворків, таких як Laravel, Symfony і CodeIgniter, які спрощують процес розробки та забезпечують такі надійні функції, як маршрутизація, абстракція бази даних і механізми шаблонів. PHP відомий простотою використання, великою документацією та великою спільнотою, яка сприяє його постійному розвитку.

Python — це універсальна мова програмування, яка славиться своєю читабельністю та простотою. Він пропонує такі потужні фреймворки, як Django та Flask, які надають повний набір інструментів для створення веб-додатків. Django наголошує на швидкій розробці, масштабованості та безпеці, що робить його придатним для великомасштабних проектів. З іншого боку, Flask пропонує легкий і гнучкий підхід, ідеальний для невеликих програм або прототипів.

Ruby разом зі своїм популярним веб-фреймворком Ruby on Rails відомий своїм елегантним і виразним синтаксисом. Ruby on Rails дотримується принципу конвенції над конфігурацією, пропонуючи структуру, яка підкреслює продуктивність і простоту розробника. Він надає такі функції, як ActiveRecord для керування базами даних і каркас для швидкого прототипування. Ruby on

					ІАЛЦ.467800.003 ПЗ	Арк.
						29
Зм.	Арк.	№ докум.	Підпис	Дата		

Rails є фаворитом серед стартапів і розробників, які прагнуть створювати програми швидко та ефективно.

Java — це надійна та широко розповсюджена мова програмування з сильною присутністю в корпоративній веб-розробці. Екосистема Java включає такі фреймворки, як Spring і JavaServer Faces (JSF), які пропонують комплексні інструменти для створення масштабованих і високопродуктивних веб-додатків. Spring фокусується на впровадженні залежностей, аспектно-орієнтованому програмуванні та модульному дизайні, тоді як JSF забезпечує компонентний підхід для створення динамічних інтерфейсів користувача.

C# — це потужна мова програмування, розроблена Microsoft, яка в основному використовується з інфраструктурою ASP.NET. ASP.NET надає багатий набір інструментів і бібліотек для веб-розробки, що дозволяє розробникам створювати безпечні високопродуктивні програми. ASP.NET наголошує на повній інтеграції з іншими технологіями Microsoft, що робить його популярним вибором для підприємств і розробників, які використовують екосистему Microsoft.

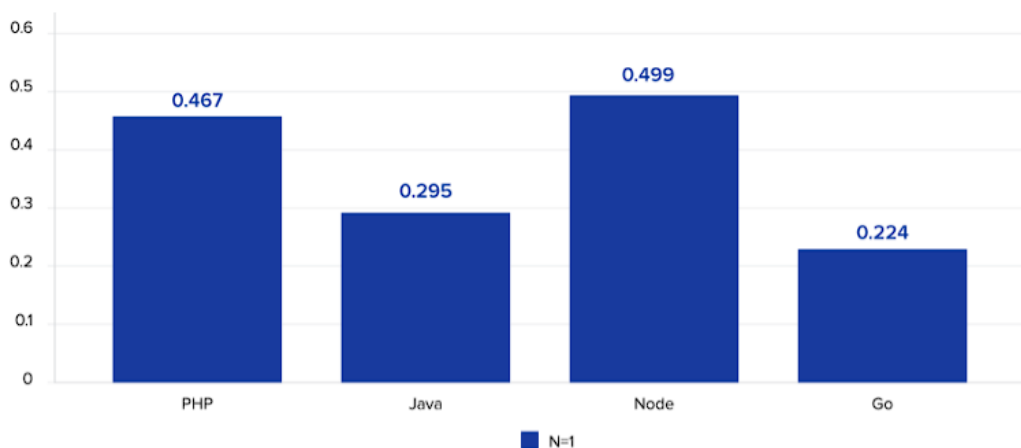


Рисунок 2.2 – Порівняльний графік швидкості серверів написаних на PHP, Java, Node, Go

Підсумовуючи, вибір мови програмування та фреймворку залежить від різних факторів, включаючи вимоги до проекту, досвід розробника та потреби в масштабованості. PHP і Python відрізняються простотою використання та

широкою підтримкою спільноти. Ruby on Rails надає перевагу простоті та швидкому розвитку. Java і C# популярні в корпоративних налаштуваннях, тоді як JavaScript із Node.js забезпечує безперебійну роботу повного стека JavaScript. Зрештою, вибір має ґрунтуватися на конкретних потребах проекту та обізнаності команди розробників. Враховуючи ці фактори, розробники можуть приймати зважені рішення та використовувати сильні сторони кожної мови програмування та фреймворку для створення потужних, масштабованих та ефективних веб-додатків.

2.4 Розгортання веб-сервера

Розгортання веб-сервера стосується процесу налаштування веб-сервера та надання доступу користувачам до нього. Це передбачає налаштування серверного середовища, установку та налаштування необхідних програмних компонентів, а також забезпечення належного підключення для обслуговування веб-додатків і вмісту. Процес розгортання може відрізнятися залежно від конкретних вимог до веб-додатку та інфраструктури, на якій він буде розміщений.

Першим кроком у розгортанні веб-сервера є визначення вимог до інфраструктури. Це включає вибір середовища розміщення, наприклад локальних серверів, хмарних платформ (наприклад, Amazon Web Services, Microsoft Azure) або керованих служб хостингу. Фактори, які слід враховувати, включають масштабованість, надійність, продуктивність, безпеку та вартість [15].

Після створення інфраструктури необхідно налаштувати сервер. Це передбачає інсталяцію операційної системи, програмного забезпечення веб-сервера (наприклад, HTTP-сервер Apache, Nginx) та інших необхідних компонентів програмного забезпечення (наприклад, системи керування базами

					ІАЛЦ.467800.003 ПЗ	Арк.
						31
Зм.	Арк.	№ докум.	Підпис	Дата		

даних, середовища виконання мов програмування). Параметри конфігурації налаштовані для оптимізації продуктивності, безпеки та використання ресурсів.

Безпека є критично важливим аспектом розгортання веб-сервера. Заходи безпеки включають налаштування брандмауерів, налаштування контролю доступу, увімкнення шифрування (наприклад, SSL/TLS) і впровадження безпечних механізмів автентифікації. Регулярно застосовуються виправлення безпеки та оновлення, щоб захистити сервер від уразливостей.

Конфігурація мережі передбачає налаштування підключення до мережі для веб-сервера. Це включає налаштування IP-адрес, параметрів DNS (системи доменних імен) і правил маршрутизації. Балансувальники навантаження та мережі доставки вмісту (CDN) також можуть бути налаштовані для підвищення продуктивності та обробки великих обсягів трафіку.

Після того, як сервер підготовлено, веб-додаток розгортається на сервері. Це передбачає передачу необхідних файлів, зокрема HTML, CSS, JavaScript, серверного коду та сценаріїв бази даних. Потім програму налаштовують для роботи в серверному середовищі з установленими всіма необхідними залежностями.

Після розгортання проводиться ретельне тестування, щоб переконатися, що веб-сервер і програма функціонують правильно. Це включає функціональне тестування, тестування продуктивності та тестування безпеки. Інструменти моніторингу налаштовані для відстеження показників сервера, виявлення проблем і забезпечення оптимальної продуктивності. Регулярний моніторинг допомагає виявити та усунути будь-які потенційні вузькі місця або зниження продуктивності [16].

Зі збільшенням веб-трафіку може виникнути необхідність масштабувати середовище веб-сервера. Це може передбачати додавання додаткових екземплярів сервера, використання балансувальників навантаження для розподілу трафіку та впровадження методів кластеризації або контейнеризації для високої доступності та відмовостійкості.

					ІАЛЦ.467800.003 ПЗ	Арк.
						32
Зм.	Арк.	№ докум.	Підпис	Дата		

Розгортання веб-сервера не є одноразовим процесом. Регулярне технічне обслуговування має важливе значення для забезпечення актуальності та безпеки сервера та програм. Це включає застосування патчів безпеки, оновлення версій програмного забезпечення та моніторинг журналів на наявність аномалій. Резервне копіювання та плани аварійного відновлення також створені для мінімізації втрати даних і забезпечення безперервності бізнесу.

2.5 Інтеграція бази даних

Інтеграція бази даних є критично важливим аспектом розробки веб-сервера, який дозволяє веб-додаткам ефективно зберігати, отримувати та маніпулювати даними. Інтеграція баз даних із веб-серверами забезпечує основу для динамічних і керованих даними веб-додатків. Ця контрольна робота досліджує важливість інтеграції бази даних у веб-сервери, різні використовувані системи керування базами даних (СУБД) і методи, які використовуються для безпроблемного підключення веб-додатків до баз даних. Розуміння цих концепцій має важливе значення для початківців веб-розробників для створення надійних і масштабованих веб-додатків:

1. Інтеграція бази даних дозволяє веб-додаткам зберігати, отримувати та маніпулювати структурованими даними. Це дозволяє користувачам взаємодіяти з додатком, надсилаючи та одержуючи інформацію. Бази даних забезпечують структурований і організований підхід до керування даними, дозволяючи ефективно зберігання, пошук і підтримку цілісності даних.
2. Для інтеграції веб-сервера доступно кілька варіантів СУБД, кожен з яких має свої функції та переваги. Деякі популярні варіанти включають реляційні бази даних, такі як MySQL, PostgreSQL і Oracle, а також бази даних NoSQL, такі як MongoDB і Redis. Вибір СУБД залежить від таких

факторів, як структура даних, масштабованість, продуктивність і специфічні вимоги веб-додатку.

3. Інтеграція бази даних відіграє важливу роль у досягненні масштабованості та продуктивності веб-додатків. Бази даних пропонують механізми горизонтального та вертикального масштабування, що дозволяє веб-серверам обробляти зростаючі обсяги даних і трафік користувачів. За допомогою таких методів, як шардинг бази даних, реплікація та кешування, веб-програми можуть розподіляти робоче навантаження та покращувати час відповіді. Крім того, оптимізовані плани виконання запитів, індексація бази даних і методи денормалізації даних можуть підвищити продуктивність операцій пошуку та обробки даних.
4. Інтеграція баз даних із веб-серверами відкриває можливості для аналізу даних і звітності. Використовуючи збережені дані, веб-додатки можуть генерувати статистичні дані, створювати звіти та приймати рішення на основі даних. Інструменти бізнес-аналітики та інфраструктури аналізу даних можна інтегрувати з базами даних, щоб отримувати значущу інформацію із зібраних даних, дозволяючи організаціям отримувати цінну інформацію та здійснювати стратегічні ініціативи.
5. Інтеграція бази даних спрощує процеси розробки та обслуговування програм. Використовуючи фреймворки, такі як ORM (Object-Relational Mapping), розробники можуть взаємодіяти з базами даних за допомогою конструкцій програмування високого рівня, зменшуючи потребу в ручних запитах SQL і забезпечуючи більш інтуїтивно зрозумілий і послідовний досвід розробки. Крім того, рівні абстракції бази даних допомагають відокремити логіку програми від основної бази даних, полегшуючи перемикання між різними параметрами СУБД або зміну схеми бази даних без впливу на всю програму.

					ІАЛЦ.467800.003 ПЗ	Арк.
						34
Зм.	Арк.	№ докум.	Підпис	Дата		

Інтеграція бази даних є важливим компонентом розробки веб-сервера, що дозволяє веб-додаткам ефективно зберігати та отримувати дані. Розуміння важливості інтеграції бази даних, різноманітних доступних варіантів СУБД і методів, які використовуються для з'єднання веб-серверів із базами даних, є важливим для створення надійних і масштабованих веб-додатків. Освоївши ці концепції, веб-розробники можуть створювати динамічні веб-програми, керовані даними, які забезпечують користувачам безперебійну взаємодію та ефективне керування даними.

2.6 Кешування та оптимізація продуктивності

Кешування та оптимізація продуктивності є ключовими аспектами розробки веб-серверів, які спрямовані на підвищення швидкості, оперативності та загальної продуктивності веб-додатків. Впроваджуючи механізми кешування та використовуючи методи оптимізації продуктивності, розробники можуть значно покращити взаємодію з користувачем і мінімізувати навантаження на сервер. Кешування передбачає зберігання даних, до яких часто звертаються, або оброблених результатів у тимчасовому сховищі, такому як пам'ять або диск, щоб уникнути надмірної обробки. Кешуючи дані, веб-сервери можуть швидше отримувати інформацію, скорочуючи час відповіді на наступні запити [17]. Кешування можна застосовувати на різних рівнях, зокрема:

1. Кешування бази даних: кешування результатів запитів до бази даних може зменшити накладні витрати на повторне виконання складних запитів. Такі інструменти, як Memcached і Redis, забезпечують кешування в пам'яті для результатів запитів до бази даних, покращуючи час відповіді.

2. Кешування вмісту: кешування статичного вмісту, як-от HTML, CSS, JavaScript і зображень, на рівні веб-сервера або проксі-сервера може звести до мінімуму потребу в частих операціях введення-виведення диска та прискорити доставку вмісту користувачам. Мережі доставки вмісту (CDN) також

використовують кешування для зберігання статичного вмісту на територіально розподілених серверах, зменшуючи затримку та покращуючи доступність вмісту.

3. Кешування на рівні програми: специфічні для програми дані, такі як сеанси користувача, обчислені значення або дані, до яких часто звертаються, можна кешувати в пам'яті. Це зменшує необхідність повторного обчислення або отримання даних із серверної частини, покращуючи продуктивність програми (рис 2.3).

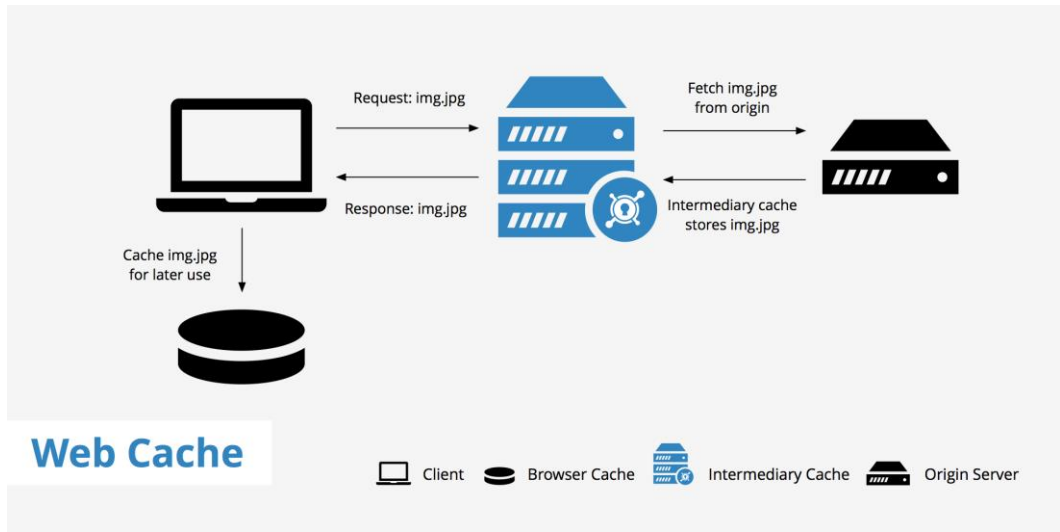


Рисунок 2.3 – Схема кешування контенту

Оптимізація продуктивності є критично важливим аспектом розробки веб-сервера, який зосереджується на покращенні швидкості, оперативності та загальної продуктивності веб-додатків. Впроваджуючи різні методи, розробники можуть покращити роботу користувача, зменшити навантаження на сервер і забезпечити ефективне використання ресурсів. Ось деякі методи оптимізації продуктивності, які зазвичай використовуються при розробці веб-серверів.

Мінімізація передбачає видалення непотрібних символів, пробілів і розривів рядків із файлів коду, таких як JavaScript і CSS, щоб зменшити їх розмір. Цей метод оптимізації покращує швидкість завантаження цих файлів, що призводить до швидшого часу завантаження сторінки. Крім того, стиснення

файлів за допомогою таких методів, як Gzip або Brotli, ще більше зменшує їхній розмір для ефективної передачі через мережу.

Налаштування відповідних заголовків кешу та політик терміну дії дозволяє веб-переглядачам зберігати статичні ресурси локально. Кешуючи такі ресурси, як зображення, CSS і файли JavaScript, наступне завантаження сторінок може бути швидшим, оскільки браузер може отримати ці файли зі свого локального кешу замість того, щоб робити нові запити до сервера.

Балансування навантаження передбачає розподіл вхідних запитів між кількома примірниками сервера для рівномірного розподілу робочого навантаження. Ця техніка забезпечує оптимальне використання ресурсів і може покращити час відповіді, запобігаючи перевантаженню будь-якого окремого сервера запитами. Балансувальники навантаження можуть бути реалізовані на апаратному або програмному рівні, і вони відіграють вирішальну роль у масштабуванні веб-додатків.

CDN — це мережі територіально розподілених серверів, які зберігають кешовані копії статичного вмісту. Розвантажуючи доставку статичного вмісту на ці сервери, CDN зменшують затримку та перевантаження мережі, що призводить до швидшої доставки вмісту користувачам. CDN також покращують доступність і масштабованість веб-додатків, особливо для користувачів, розташованих далеко від вихідного сервера.

2.7 Масштабованість і висока доступність

Масштабованість і висока доступність є критично важливими міркуваннями при розробці веб-серверів, щоб гарантувати, що веб-додатки можуть справлятися зі зростаючими навантаженнями та надавати безперебійне обслуговування користувачам. Масштабованість означає здатність веб-сервера обробляти зростаючі обсяги трафіку, даних і запитів користувачів без втрати продуктивності. З іншого боку, висока доступність спрямована на мінімізацію

					ІАЛЦ.467800.003 ПЗ	Арк.
						37
Зм.	Арк.	№ докум.	Підпис	Дата		

часу простою та забезпечення того, що веб-додаток залишається доступним навіть у разі збоїв або збоїв [18].

Масштабованість необхідна для адаптації до зростання веб-додатків і обробки зростаючих навантажень. Існує два основних типи масштабованості: вертикальна масштабованість і горизонтальна масштабованість. Вертикальна масштабованість передбачає додавання додаткових ресурсів до одного сервера, наприклад збільшення обчислювальної потужності сервера, пам'яті або ємності зберігання (рис 2.4). Хоча вертикальна масштабованість забезпечує простий підхід, вона має обмеження щодо максимальної потужності, яку може обслуговувати один сервер.

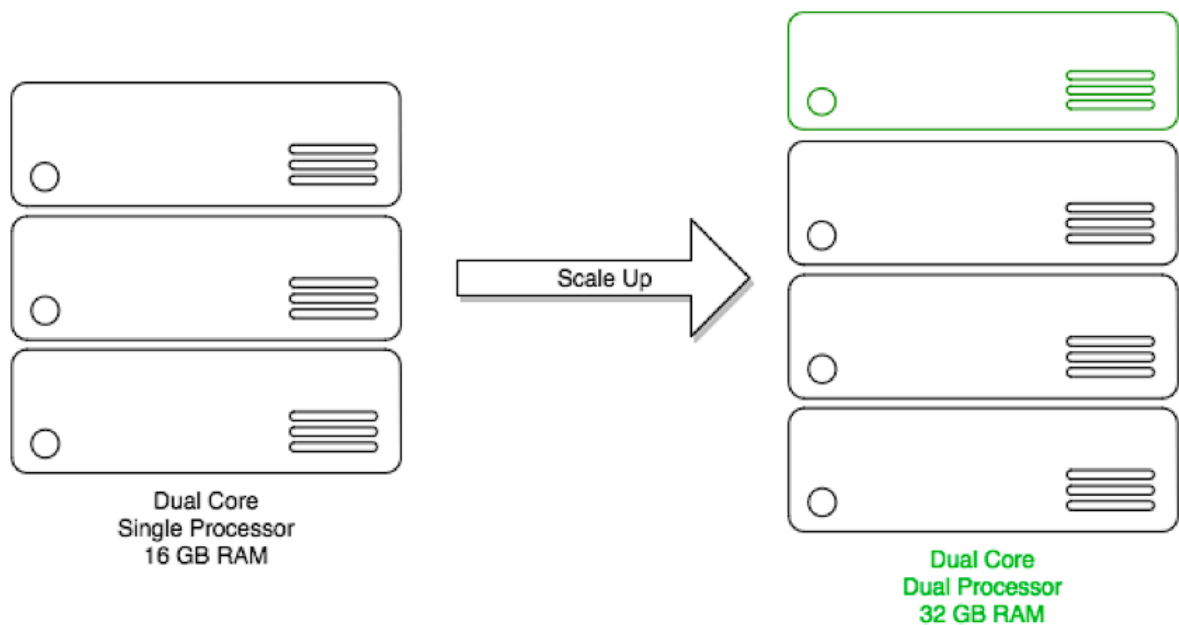


Рисунок 2.4 – Приклад вертикального масштабування системи

Горизонтальна масштабованість, з іншого боку, передбачає розподіл робочого навантаження між декількома серверами. Додаючи більше серверів до інфраструктури та розподіляючи навантаження, горизонтальна масштабованість дозволяє краще використовувати ресурси та збільшити пропускну здатність. Для рівномірного розподілу вхідних запитів між цими серверами можна використовувати такі методи балансування навантаження, як циклічний, найменші з'єднання або зважений розподіл (рис 2.5). Це гарантує,

що програма може обробляти більші обсяги трафіку, підтримувати оптимальну продуктивність і уникати надмірного навантаження на один сервер.

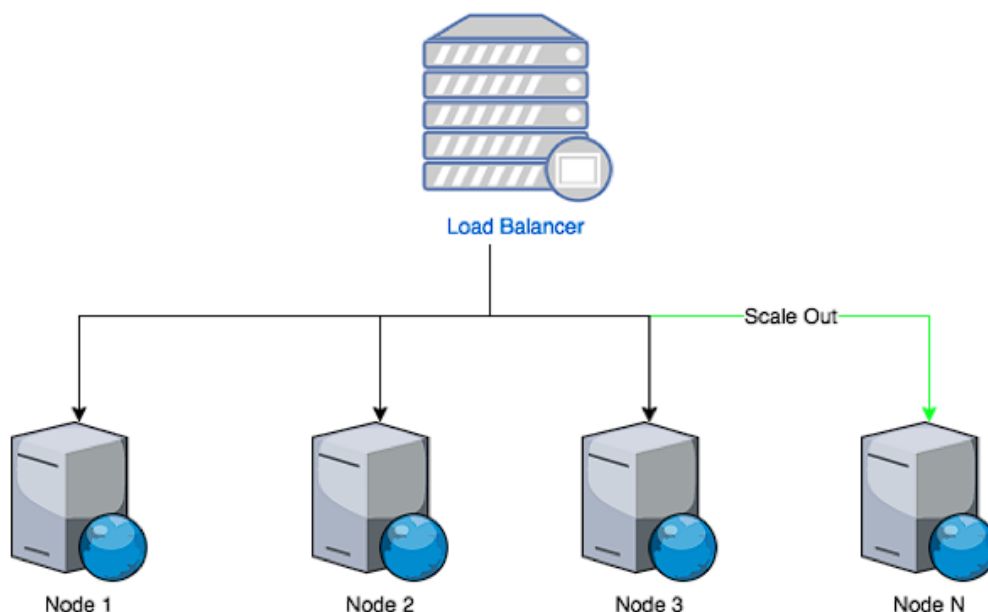


Рисунок 2.5 – Приклад горизонтального масштабування системи

Висока доступність має вирішальне значення для веб-серверів, щоб забезпечити безперебійне обслуговування користувачів. Це передбачає впровадження заходів для пом'якшення збоїв, мінімізації часу простою та підтримки доступності веб-додатку. Одним із поширених підходів до досягнення високої доступності є резервування. Завдяки дублюванню критичних компонентів, таких як сервери, бази даних або мережева інфраструктура, резервування гарантує, що в разі збою одного компонента будуть створені резервні копії або механізми відновлення після збоїв для плавного переходу до системи резервного копіювання.

Балансувальники навантаження, згадані раніше в масштабованості, також відіграють важливу роль у високій доступності. Розподіляючи трафік між декількома серверами, балансувальники навантаження можуть виявити, якщо сервер перестає відповідати або виходить з ладу, і перенаправляти трафік на справні сервери. Це дозволяє веб-програмі продовжувати працювати, навіть якщо на одному або кількох серверах виникають проблеми. Крім того, можна

використовувати такі технології, як кластеризація, реплікація та синхронізація даних, щоб забезпечити узгодженість даних у резервованих системах.

Крім того, впровадження ефективних систем моніторингу та оповіщення має вирішальне значення для високої доступності. Постійний моніторинг дозволяє в режимі реального часу бачити стан і продуктивність інфраструктури веб-сервера. У разі виникнення будь-яких проблем, таких як високе використання процесора, обмеження пам'яті або вузькі місця в мережі, адміністратори можуть бути негайно повідомлені, щоб вжити заходів для виправлення та запобігти потенційному простою.

Підсумовуючи, масштабованість і висока доступність є важливими міркуваннями при розробці веб-сервера. Масштабованість гарантує, що веб-додаток може обробляти зростаючий трафік і зростаючі вимоги, а висока доступність мінімізує час простою та забезпечує безперебійне обслуговування користувачів. Використовуючи такі методи, як балансування навантаження, резервування, кластеризація та ефективний моніторинг, розробники веб-серверів можуть створити надійну та стійку інфраструктуру, яка може масштабуватися відповідно до вимог користувачів і підтримувати постійну доступність.

2.8 Нові тенденції та технології у створенні веб-серверів

Сфера розробки веб-серверів постійно розвивається під впливом нових тенденцій і технологій, спрямованих на підвищення продуктивності, безпеки, масштабованості та взаємодії з користувачем. Ось кілька важливих тенденцій і технологій, які формують майбутнє розробки веб-серверів:

1. Архітектура мікросервісів передбачає розбиття веб-додатків на менші, слабко пов'язані служби, які можна розробляти, розгортати та масштабувати незалежно. Кожен мікросервіс зосереджений на певних бізнес-можливостях, що забезпечує більшу гнучкість,

					ІАЛЦ.467800.003 ПЗ	Арк.
						40
Зм.	Арк.	№ докум.	Підпис	Дата		

масштабованість і простоту обслуговування. Використовуючи такі технології, як контейнеризація (наприклад, Docker) і фреймворки оркестровки (наприклад, Kubernetes), веб-сервери можуть ефективно керувати та розгортати мікросервіси, що призводить до покращеної масштабованості та відмовостійкості.

2. GraphQL — це альтернативний підхід до традиційних RESTful API, який забезпечує більшу гнучкість і ефективність у пошуку даних. Це дозволяє клієнтам запитувати певні вимоги до даних, дозволяючи їм отримати лише необхідну інформацію в одному запиті. GraphQL пропонує більш оптимізований і детальний підхід до отримання даних, що призводить до зменшення надмірної та недобірки, покращення продуктивності та підвищення продуктивності розробника [19].
3. WebSockets — це технологія, яка забезпечує двонаправлений зв'язок у реальному часі між веб-серверами та клієнтами. На відміну від традиційного HTTP, який працює за моделлю запит-відповідь, WebSockets встановлює постійне з'єднання, дозволяючи серверам надсилати дані клієнтам і забезпечуючи оновлення в реальному часі без необхідності частих запитів. WebSockets особливо корисні в програмах, які вимагають миттєвих оновлень, таких як програми для чату, інструменти для спільної роботи та інформаційні панелі в режимі реального часу.
4. Технології штучного інтелекту (AI) і машинного навчання (ML) все частіше інтегруються в розробку веб-серверів. Алгоритми штучного інтелекту та машинного навчання можуть аналізувати поведінку користувачів, шаблони та дані, щоб персоналізувати вміст, покращити функції пошуку, оптимізувати рекомендації та автоматизувати процеси прийняття рішень. Веб-сервери використовують технології штучного інтелекту та машинного навчання, щоб надавати користувачам більш адаптований та інтелектуальний досвід,

					ІАЛЦ.467800.003 ПЗ	Арк.
						41
Зм.	Арк.	№ докум.	Підпис	Дата		

підвищувати ефективність і оптимізувати розподіл ресурсів [20]. Чат-боти та віртуальні помічники на основі штучного інтелекту зазвичай інтегруються у веб-сайти, щоб забезпечити миттєву підтримку клієнтів і покращити залучення користувачів. Ці боти використовують методи обробки природної мови (NLP) і алгоритми ML, щоб розуміти запити користувачів, надавати відповідні відповіді та виконувати такі завдання, як бронювання зустрічей або надання рекомендацій. ML може автоматизувати процеси тестування та налагодження веб-розробки. Інструменти тестування на основі штучного інтелекту можуть симулювати взаємодію користувача, виявляти помилки та створювати тестові випадки, зменшуючи час і зусилля, необхідні для ручного тестування.

Ці нові тенденції та технології змінюють ландшафт розробки веб-серверів, стимулюючи прогрес у масштабованості, продуктивності, спілкуванні в реальному часі та інтелектуальній обробці даних. Враховуючи ці тенденції та впроваджуючи відповідні технології, розробники веб-серверів можуть створювати більш надійні, ефективні та орієнтовані на користувача веб-додатки, які відповідають мінливим вимогам цифрового світу.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі було представлено огляд фундаментальних принципів побудови веб-серверів, досліджено різні мови програмування та фреймворки, які зазвичай використовуються для їх побудови, обговорено процес розгортання веб-сервера, заглиблено в важливість кешування та оптимізації продуктивності, а також висвітлив нові тенденції та технології у створенні веб-серверів.

Ми почали з вивчення фундаментальної архітектури веб-серверів, включаючи модель клієнт-сервер, протокол HTTP та цикл запит-відповідь. Розуміння цих концепцій має вирішальне значення для розробки ефективних і надійних програм веб-сервера.

Далі ми досліджували різні мови програмування та фреймворки, які використовуються в розробці веб-серверів, наприклад Python з Django, Ruby з Ruby on Rails і JavaScript з Node.js. Кожна мова та фреймворк мають свої сильні сторони та можуть бути обрані на основі таких факторів, як досвід розробника, вимоги до проекту та міркування продуктивності.

У розділі також обговорювалося розгортання веб-сервера, включаючи міркування щодо вибору відповідного середовища хостингу, конфігурації сервера та заходів безпеки. Ефективне розгортання веб-серверів має важливе значення для забезпечення доступності, масштабованості та продуктивності.

Кешування та оптимізація продуктивності розглядалися як важливі методи підвищення продуктивності веб-сервера. Ми обговорили різні механізми кешування, включаючи кешування бази даних, кешування вмісту та кешування на рівні програми. Крім того, для покращення часу відгуку та загальної ефективності було виділено методи оптимізації продуктивності, такі як мінімізація, стиснення, балансування навантаження та асинхронна обробка.

Нарешті, ми торкнулися нових тенденцій і технологій у створенні веб-серверів, включаючи безсерверну архітектуру, мікросервіси, GraphQL,

					ІАЛЦ.467800.003 ПЗ	Арк.
						43
Зм.	Арк.	№ докум.	Підпис	Дата		

WebSockets, периферійні обчислення та інтеграцію AI/ML. Ці досягнення формують майбутнє розробки веб-серверів, забезпечуючи більшу масштабованість, зв'язок у реальному часі, інтелектуальну обробку даних і покращений досвід користувачів.

Розуміючи та застосовуючи принципи побудови веб-серверів, використовуючи відповідні мови програмування та фреймворки, оптимізуючи продуктивність за допомогою кешування та йдучи в ногу з новими тенденціями та технологіями, розробники можуть створювати високопродуктивні, масштабовані та інноваційні додатки для веб-серверів, які відповідають вимогам, що розвиваються. вимоги сучасної веб-розробки.

Загалом, цей розділ забезпечує міцну основу для подальшого дослідження та аналізу розробки веб-серверів, закладаючи основу для наступних розділів і сприяючи всебічному розумінню створення ефективних і надійних веб-серверів.

					ІАЛЦ.467800.003 ПЗ	Арк.
						44
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ОГЛЯД РОЗРОБЛЕНОЇ СИСТЕМИ

3.1 Визначення

У сучасному бізнес-середовищі, що швидко розвивається, ефективне управління розробкою проектів має важливе значення для того, щоб організації успішно реалізовували проекти вчасно та в рамках бюджету. Веб-додатки стали потужними інструментами для управління проектами, пропонуючи централізовану платформу для співпраці, спілкування та керування завданнями. Ця стаття має на меті забезпечити поглиблене дослідження розробки клієнта та сервера для веб-додатку керування розробкою проекту, зосереджуючись на ключових компонентах і міркуваннях, пов'язаних із створенням такої системи.

Розробка веб-додатку для керування розробкою проекту включає два важливі аспекти: розробку на стороні клієнта та розробку на стороні сервера. Розробка на стороні клієнта охоплює інтерфейс користувача, інтерактивність і взаємодію з користувачем веб-додатку, тоді як розробка на стороні сервера зосереджується на логіці серверної частини, управлінні даними та спілкуванні з клієнтом.

Розробка на стороні клієнта відіграє ключову роль у забезпеченні бездоганного та інтуїтивно зрозумілого досвіду користувача. Він передбачає розробку та впровадження інтерфейсу користувача, який включає такі функції, як створення проекту, призначення завдань, відстеження прогресу, обмін документами та командна співпраця. Користувальницький інтерфейс має бути візуально привабливим, зручним і чуйним, дозволяючи користувачам ефективно переміщатися програмою та без зусиль виконувати завдання з керування проектами. Для розробки на стороні клієнта зазвичай використовуються такі технології, як HTML, CSS і фреймворки JavaScript, як-от React, Angular або Vue.js.

					ІАЛЦ.467800.003 ПЗ	Арк.
						45
Зм.	Арк.	№ докум.	Підпис	Дата		

З іншого боку, розробка на стороні сервера є основою веб-додатку для керування розробкою проекту. Він передбачає обробку запитів користувачів, керування зберіганням і пошуком даних, забезпечення заходів безпеки та полегшення зв'язку між клієнтом і сервером. Розробка на стороні сервера зазвичай передбачає вибір відповідної мови програмування (наприклад, Python, Ruby або JavaScript), вибір веб-фреймворку (наприклад, Django, Ruby on Rails або Node.js) і розробку надійної моделі даних і схеми бази даних зберігати та керувати інформацією про проект. Крім того, впровадження механізмів автентифікації та контролю доступу для забезпечення безпеки даних і конфіденційності користувачів є критичним.

Крім того, розробка на стороні сервера також включає інтеграцію сторонніх API або служб для додаткових функцій, таких як повідомлення електронною поштою, зберігання файлів або інтеграція з іншими інструментами управління проектами. Ця інтеграція забезпечує безперебійну співпрацю та підвищує загальну ефективність процесу управління розробкою проекту.

3.2 Обґрунтування обраних засобів розробки – React

React, популярна бібліотека JavaScript для створення користувацьких інтерфейсів, пропонує кілька переваг для розробки веб-додатків для керування розробкою проектів:

1. Компонентна архітектура React дозволяє створювати багаторазові компоненти інтерфейсу користувача. Ця модульність дає змогу створювати послідовний і візуально привабливий інтерфейс користувача, що забезпечує професійну та зручну роботу для керівників проектів, членів команди та зацікавлених сторін. Декларативний синтаксис React також полегшує розуміння та

					ІАЛЦ.467800.003 ПЗ	Арк.
						46
Зм.	Арк.	№ докум.	Підпис	Дата		

оновлення компонентів інтерфейсу користувача, підвищуючи ефективність розробки та обслуговування.

2. React використовує віртуальну DOM, яка дозволяє ефективно оновлювати та відтворювати лише необхідні компоненти, коли є зміни в стані програми. Такий підхід мінімізує непотрібне повторне відтворення, що призводить до підвищення продуктивності та швидшого оновлення інтерфейсу користувача. У веб-додатку для керування розробкою проектів, де спільна робота в режимі реального часу й оновлення даних є звичайним явищем, ефективний рендеринг React сприяє плавному та швидкому реагуванню користувачів.
3. Односпрямований потік даних React і рішення для керування станом, такі як Redux або MobX, спрощують обробку складного стану програми. У веб-додатку для керування розробкою проекту, де кілька користувачів взаємодіють із завданнями, часовими шкалами та даними проекту, ефективне керування станом має вирішальне значення. Рішення React для керування станом забезпечують централізований підхід до керування та синхронізації даних, зменшуючи складність і потенційну невідповідність даних.
4. React отримує переваги від великої й активної спільноти розробників, що створює багату екосистему бібліотек, інструментів і ресурсів. Ця екосистема пропонує широкий спектр готових компонентів, бібліотек інтерфейсу користувача та інструментів розробки, які можуть прискорити процес розробки та покращити функціональність вашої веб-програми для керування розробкою проектів. Потужна підтримка спільноти забезпечує регулярне оновлення, виправлення помилок і доступ до цінних ресурсів і досвіду.

Таким чином, використання React як бібліотеки для розробки веб-додатків для керування розробкою проектів пропонує такі переваги, як покращений інтерфейс користувача, ефективне відтворення та продуктивність,

					ІАЛЦ.467800.003 ПЗ	Арк.
						47
Зм.	Арк.	№ докум.	Підпис	Дата		

безперебійне керування станом, доступ до надійної екосистеми та крос-платформна сумісність. Ці переваги сприяють розробці масштабованої, інтерактивної та зручної програми, яка дає змогу командам проектів ефективно керувати проектами, ефективно співпрацювати та досягати успішних результатів.

3.3 Обґрунтування обраних засобів розробки – Express

Express — це мінімалістичний веб-фреймворк для Node.js, який забезпечує основні функції для створення веб-додатків без зайвих витрат. Він зосереджений на простоті та гнучкості, дозволяючи розробляти та структурувати програму керування розробкою проектів відповідно до ваших конкретних вимог. Ця легка природа робить Express ідеальним вибором, якщо ви хочете зберегти свою програму оптимізованою та ефективною (рис 3.1).

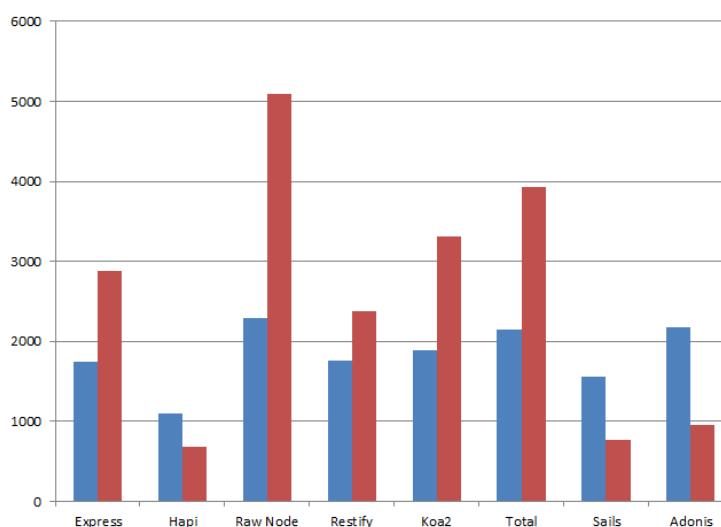


Рисунок 3.1 – Порівняння швидкості роботи веб-серверів, що написані за допомогою різних бібліотек

Express пропонує потужну систему маршрутизації, яка дозволяє визначати маршрути для різних кінцевих точок і методів HTTP. Це дає змогу створити добре організований і структурований API для програми керування розробкою проекту. Express також надає систему проміжного програмного забезпечення, яка дозволяє додавати спеціальну логіку, таку як автентифікація,

розбір запитів, обробка помилок і журналювання, на різних етапах циклу запит-відповідь. Система проміжного програмного забезпечення покращує функціональність і розширюваність програми, полегшуючи виконання складних завдань та інтеграцію з іншими компонентами.

Express надає розробникам високий рівень гнучкості, дозволяючи налаштовувати та розширювати структуру відповідно до потреб. Є можливість використовувати численні модулі проміжного програмного забезпечення сторонніх розробників, доступні в екосистемі npm, щоб покращити програму керування розробкою проєктів додатковими функціями та функціями. Модульна природа Express дозволяє вибирати та інтегрувати лише ті компоненти та залежності, які мають відношення до програми, створюючи індивідуальне та оптимізоване рішення.

Express відомий своєю чудовою продуктивністю та масштабованістю. Він має неблокуючу архітектуру, керовану подіями, що робить його добре придатним для обробки одночасних запитів і масштабування відповідно до потреб зростаючої бази користувачів. Express також надає такі функції, як кластеризація, що дозволяє використовувати переваги багатоядерних систем і ефективно розподіляти робоче навантаження. Ця масштабованість і продуктивність дозволяють ефективно обробляти велику кількість користувачів, складні операції з даними та оновлення в реальному часі.

3.4 Розробка серверної частини

3.4.1 Авторизація та реєстрація

Реалізація авторизації та автентифікації має вирішальне значення для захисту веб-програми, створеної за допомогою Express. Ці механізми допомагають контролювати доступ до певних ресурсів, перевіряти особу користувачів і забезпечувати конфіденційність і цілісність конфіденційних

					ІАЛЦ.467800.003 ПЗ	Арк.
						49
Зм.	Арк.	№ докум.	Підпис	Дата		

даних. Ось кілька ключових міркувань щодо впровадження авторизації та автентифікації в Express.

Автентифікація — це процес підтвердження особи користувача. Це гарантує, що лише автентифіковані користувачі можуть отримати доступ до захищених ресурсів. Express надає різноманітне проміжне програмне забезпечення та бібліотеки, які полегшують реалізацію автентифікації. Однією з популярних бібліотек є Passport.js, яка пропонує гнучкий і модульний підхід до автентифікації. Passport.js підтримує кілька стратегій автентифікації, таких як локальне ім'я користувача/пароль, соціальні входи (OAuth), веб-токени JSON (JWT) тощо. За допомогою Passport.js ви можете легко інтегрувати механізми автентифікації і керувати входом користувача, реєстрацією та керуванням сесіями.

Авторизація визначає, до яких дій або ресурсів користувач має доступ у додатку. Це важливо для забезпечення контролю доступу на основі ролей і захисту конфіденційних функцій. Express не має вбудованих механізмів авторизації, але можна реалізувати це за допомогою проміжного програмного забезпечення та спеціальної логіки. Зберігаючи ролі користувачів або дозволи у моделі користувача або базі даних, можна створювати функції проміжного програмного забезпечення, які перевірятимуть, чи має користувач необхідні привілеї для доступу до певних маршрутів або виконання певних дій. Ці функції проміжного програмного забезпечення можна застосовувати вибірково для захисту маршрутів або кінцевих точок, гарантуючи, що лише авторизовані користувачі можуть виконувати заплановані дії.

JWT — це широко використовуваний механізм автентифікації для веб-додатків. Це компактний самодостатній маркер, який містить претензії щодо користувача та може використовуватися для автентифікації та авторизації. Express може легко інтегрувати автентифікацію JWT за допомогою проміжного програмного забезпечення, такого як jsonwebtoken. Коли користувач успішно ввійшов у систему, ви можете надіслати JWT клієнту, який буде включено в

					ІАЛЦ.467800.003 ПЗ	Арк.
						50
Зм.	Арк.	№ докум.	Підпис	Дата		

наступні запити як заголовок авторизації. На стороні сервера ви можете перевірити та декодувати JWT, витягнувши ідентифікатор користувача та іншу відповідну інформацію для виконання перевірок авторизації. JWT забезпечують автентифікацію без збереження стану, усуваючи необхідність зберігати дані сеансу на сервері.

Для реалізації реєстрації користувача в додатку, в першу чергу повинні бути створені моделі сутностей, що будуть зберігати данні про користувачів в базі даних. На рис 3.2 зображення модель, що створена за допомогою класу з бібліотеки mongoose. Тобто об'єкт користувача буде містити такі поля як пошта, прізвище, ім'я, пароль, а також роль користувача (яка буде зберігати id об'єкту, що міститься в колекції Roles). Всі поля визначені як обов'язкові, крім ролі користувача, проте за замовчуванням, якщо користувач не вказав конкретної ролі, йому буде видана роль розробника. JWT використовуються для безпечної передачі інформації між сторонами як об'єкт JSON. Вони мають цифровий підпис і можуть бути зашифровані, забезпечуючи безпечний спосіб передачі даних. JWT складається з трьох частин: заголовка, корисного навантаження та підпису.

Заголовок містить інформацію про алгоритм, який використовується для підписання маркера. Корисне навантаження містить претензії або інформацію про користувача або сутність.

```
models > JS user.js > ...
1  const { Schema, model } = require('mongoose');
2
3  const User = new Schema({
4    email: { type: String, unique: true, require: true },
5    name: { type: String, require: true },
6    surname: { type: String, require: true },
7    password: { type: String, require: true },
8    roles: [{ type: String, ref: 'Role' }],
9  });
10
11 module.exports = model('User', User);
```

Рисунок 3.2 – Модель User.js для зберігання сутностей користувачів

Ці претензії можуть містити такі речі, як ідентифікатор користувача, роль, термін дії та будь-які необхідні спеціальні дані. Підпис використовується для перевірки цілісності токена та гарантії, що його не було підроблено.

Також для програмної реалізації потрібно створити модель для зберігання сутності ролі (рис 3.3), і кожен об'єкт в колекції Role буде відповідати одній з доступних ролей (тобто кількість ролей можна буде розширяти). В додатку було використано лише дві ролі: менеджер та розробник.

```
models > JS role.js > ...
1  const { Schema, model } = require('mongoose');
2
3  const Role = new Schema({
4    role: { type: String, require: true, unique: true, default: 'dev' },
5  });
6
7  module.exports = model('Role', Role);
```

Рисунок 3.3 – Модель Role.js для зберігання сутностей користувачів

Далі нам потрібно декілька методів, що будуть безпосередньо відповідати за логіку авторизації та реєстрації:

- **login:** цей метод (рис 3.4) обробляє функцію входу користувача. Він отримує об'єкти req (запит) і res (відповідь) як параметри. Він витягує адресу електронної пошти та пароль із тіла запиту. Потім він намагається знайти користувача в базі даних із наданою електронною поштою за допомогою User.findOne(). Якщо користувача не знайдено, він повертає відповідь JSON із кодом статусу 400 і повідомленням про те, що користувача не знайдено. Якщо користувача знайдено, він порівнює наданий пароль із хешованим паролем користувача за допомогою bcryptjs.compareSync(). Якщо пароль недійсний, він повертає відповідь JSON із кодом статусу 400 і повідомленням із зазначенням недійсних облікових даних. Якщо пароль дійсний, він генерує токен за допомогою функції generateToken (не показано в наданому коді) і повертає відповідь JSON із токеном. Впровадивши систему входу, ви можете захистити конфіденційні області вашої веб-програми або API від несанкціонованого доступу. Автентифікація користувачів гарантує, що лише автентифіковані користувачі можуть отримати доступ до захищених ресурсів. За допомогою системи входу ви можете надати користувачам

					ІАЛЦ.467800.003 ПЗ	Арк.
						52
Зм.	Арк.	№ докум.	Підпис	Дата		

персоналізований досвід. Після автентифікації користувачі можуть мати власні профілі, налаштування та доступ до персоналізованого вмісту чи функцій.

```

async login(req, res) {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(400).json({ message: 'User was not found' });
    }
    const isValidPassword = bcryptjs.compareSync(password, user.password);
    if (!isValidPassword) {
      return res.status(400).json({ message: 'Invalid credentials' });
    }
    const token = generateToken(user._id, user.roles);
    return res.json({ token });
  } catch (e) {
    res.status(400).json({ message: 'Error during login' });
  }
}

```

Рисунок 3.4 – Лістинг методу login для входу користувача

- **registration:** цей метод (рис. 3.5) обробляє функцію реєстрації користувача. Він отримує об'єкти req (запит) і res (відповідь) як параметри. Спочатку він перевіряє запит за допомогою validationResult (з бібліотеки express-validator). Якщо є помилки перевірки, він повертає відповідь JSON із кодом статусу 400 і повідомленням про помилку. Якщо запит проходить перевірку, він витягує електронну адресу, ім'я, прізвище, пароль і роль із тіла запиту. Він намагається знайти користувача в базі даних із наданою електронною поштою за допомогою User.findOne(). Якщо існує користувач із такою ж електронною адресою, він повертає відповідь JSON із кодом статусу 400 і повідомленням про те, що електронна адреса вже зареєстрована. Якщо не знайдено жодного користувача з такою ж електронною адресою, він хешує пароль за допомогою bcryptjs.hashSync() і створює новий об'єкт користувача. Він

зберігає користувача в базі даних за допомогою `user.save()` і повертає відповідь JSON із повідомленням про успіх.

В свою чергу, для перевірки ролей був створений `middleware` (рис 3.6), який працює наступним чином:

```
async registration(req, res) {
  try {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ message: 'Error during registration' });
    }
    const { email, name, surname, password, role } = req.body;
    const candidate = await User.findOne({ email });
    if (candidate) {
      return res.status(400).json({ message: 'Email is already registered' });
    }
    const hashedPassword = bcryptjs.hashSync(password, 7);
    const userRole = await Role.findOne({ role });
    const user = new User({ email, name, surname, password: hashedPassword, roles: [userRole.role] });
    await user.save();
    return res.json({ message: 'User has been created' });
  } catch (e) {
    res.status(400).json({ message: 'Error during registration', error: e });
  }
}
```

Рисунок 3.5 – Лістинг методу `registration` для входу користувача

1. Функція приймає три параметри: `req` (запит), `res` (відповідь) і `next` (функція зворотного виклику для передачі керування наступній функції проміжного програмного забезпечення в ланцюжку).
2. Функція спочатку перевіряє, чи є метод запиту «OPTIONS». Якщо так, то негайно викликається функція `next()`, щоб передати керування наступному проміжному ПЗ. Це зазвичай використовується для обробки запитів перед польотом у CORS (Cross-Origin Resource Sharing).
3. Якщо метод запиту не є «OPTIONS», функція намагається витягти маркер із заголовка авторизації запиту. Припускається, що токен передається в заголовку авторизації як `bearer` токен у форматі «`Bearer {token}`». Він розділяє значення заголовка на пробіл і отримує другу частину, яка має бути токеном.

4. Якщо токен не знайдено або він хибний, функція повертає відповідь JSON із кодом статусу 403 (Forbidden) і повідомленням про те, що користувач не авторизований.
5. Якщо токен знайдено, він намагається перевірити токен за допомогою `jwt.verify()`. Для процесу перевірки потрібен секретний ключ, який є секретним. Ця функція перевіряє цілісність і автентичність токена і повертає розшифроване значення.
6. Якщо токен успішно перевірено, розшифроване значення призначається `req.user`, що робить його доступним у наступних обробниках маршрутів.
7. Нарешті, функція викликає функцію `next()`, щоб передати керування наступному проміжному програмному забезпеченню або обробнику маршруту в ланцюжку.

```

middleware > JS authorization-middleware.js > ...
1  const jwt = require('jsonwebtoken');
2  const { secret } = require('../config');
3
4  module.exports = function (req, res, next) {
5    if (req.method === 'OPTIONS') {
6      next();
7    }
8
9    try {
10     const token = req.headers.authorization.split(' ')[1];
11     if (!token) {
12       return res.status(403).json({ message: 'User is not authorized' });
13     }
14     const decoded = jwt.verify(token, secret);
15     req.user = decoded;
16     next();
17   } catch (e) {
18     console.log(e);
19     res.status(403).json({ message: 'User is not authorized' });
20   }
21 };

```

Рисунок 3.6 – Лістинг методу для перевірки токена

3.4.2 Методи для керування проектами

Веб додаток для керування розробкою проектів, очевидно, повинен містити методи для створення і керування проектами. Для початку, потрібно

створити модель, що буде відповідати сутності «проект», що буде містити наступні поля:

- **title** – назва проекту
- **description** – опис проекту
- **startDate** – початковий день розробки
- **endDate** – кінцевий день розробки
- **status** – статус проекту, який може бути лише “ONGOING”, “COMPLETED” або “PAUSED”
- **tasks** – це масив посилань на задачі, що відносяться до якогось конкретного проекту
- **createdBy** – посилання на користувача, що створив проект
- **createdAt** – час, коли проект був створений

Структуру моделі можна побачити на рис. 3.7

```
const projectSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true
  },
  description: {
    type: String,
    required: true
  },
  startDate: {
    type: Date,
    required: true
  },
  endDate: {
    type: Date,
    required: true
  },
  status: {
    type: String,
    enum: ['ongoing', 'completed', 'paused'],
    default: 'ongoing'
  },
  tasks: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Task'
    }
  ],
  createdBy: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
});
```

Рисунок 3.7 – Модель Project.js для зберігання сутностей користувачів

Для того щоб створити проект, потрібно використовувати запит “project/create”, реалізація якого наведена нижче на рис. 3.8

					ІАЛЦ.467800.003 ПЗ	Арк.
						56
Зм.	Арк.	№ докум.	Підпис	Дата		

Потім ми створюємо новий екземпляр моделі Project, використовуючи витягнуті деталі проекту. Ми зберігаємо проект до бази даних за допомогою методу save(), який є асинхронною операцією.

```

1  const express = require('express');
2  const app = express();
3  const mongoose = require('mongoose');
4  const Project = require('./models/Project');
5
6
7  app.post('/create', async (req, res) => {
8    try {
9      const { title, description, startDate, endDate, createdBy } = req.body;
10
11     const project = new Project({
12       title,
13       description,
14       startDate,
15       endDate,
16       createdBy,
17     });
18
19     const savedProject = await project.save();
20     res.status(200).json({ message: 'Project created successfully', project: savedProject });
21   } catch (error) {
22     res.status(400).json({ message: 'Error creating project', error: error.message });
23   }
24 });

```

Рисунок 3.8 – Лістинг роуту «project/create»

Якщо створення проекту пройшло успішно, ми відповідаємо повідомленням про успішне завершення та деталями створеного проекту. Якщо під час процесу створення виникають будь-які помилки, ми їх обробляємо та відповідаємо відповідним повідомленням про помилку.

Схожим чином працює запит “project/:projectId/update” (рис 3.9), проте є декілька моментів, які потрібно врахувати. Ми використовуємо Project.findById(), щоб знайти проект у базі даних на основі його ідентифікатора. Якщо проект не знайдено, ми відповідаємо статусом 404 і відповідним повідомленням про помилку.

Якщо проект знайдено, ми оновлюємо його властивості новими деталями, наданими в тілі запиту. Ми також оновлюємо поле modifiedAt, щоб відстежувати мітку часу модифікації.

Нарешті, ми зберігаємо оновлений проект до бази даних за допомогою методу save(). Якщо оновлення пройшло успішно, ми відповідаємо

повідомленням про успіх і оновленими деталями проекту. Якщо під час оновлення виникають будь-які помилки, ми вирішуємо їх і відповідаємо відповідним повідомленням про помилку.

```
--
27 app.put('projects/:projectId/update', async (req, res) => {
28   try {
29     const { projectId } = req.params;
30
31     const project = await Project.findById(projectId);
32
33     if (!project) {
34       return res.status(404).json({ message: 'Project not found' });
35     }
36
37     project.title = req.body.title;
38     project.description = req.body.description;
39     project.startDate = req.body.startDate;
40     project.endDate = req.body.endDate;
41     project.modifiedAt = new Date();
42
43     const updatedProject = await project.save();
44
45     res.status(200).json({ message: 'Project updated successfully', project: updatedProject });
46   } catch (error) {
47     res.status(400).json({ message: 'Error updating project', error: error.message });
48   }
49 });
--
```

Рисунок 3.9 – Лістинг роуту «project/:projectId/update»

3.4.3 Методи для керування задачами

Наступні ендпоїнти дозволяють керувати завданнями в рамках проекту. Є можливість отримувати завдання, створювати нові завдання, оновлювати існуючі завдання та видаляти завдання. Параметр {projectId} представляє ідентифікатор проекту, пов'язаного із завданнями, тоді як параметр {taskId} представляє ідентифікатор конкретного завдання:

- GET projects/{projectId}/tasks - цей ендпоїнт дозволяє отримати список всіх завдань, пов'язаних з конкретним проектом. Запит на цей ендпоїнт поверне список завдань, включаючи їх деталі, такі як назва, опис, статус тощо.
- GET projects/{projectId}/task/{taskId} – отримати інформацію про окрему задачу відносно вибраного проекту
- POST projects/{projectId}/taskCreate – Цей ендпоїнт дозволяє створити нове завдання для вказаного проекту. В тілі запиту передаються необхідні дані для створення завдання, такі як назва, опис, призначення тощо. Після

					ІАЛЦ.467800.003 ПЗ	Арк.
						58
Зм.	Арк.	№ докум.	Підпис	Дата		

успішного створення завдання, сервер повертає статус 201 зі створеним завданням у відповіді.

- POST projects/{projectId}/taskUpdate/{taskId} – оновлення інформації щодо задачі (статус, коментарі і тд.).

В першу чергу потрібно створити модель, що буде відповідати сутності Task. На рис. 3.10 визначена схема Mongoose для об'єкта «task». Схема визначає різні властивості для завдання, зокрема назву, опис, assignedTo (що представляє особу, відповідальну за завдання), термін виконання, статус (зі значенням за замовчуванням «To Do») і проект (який є посиланням на асоційований проект, використовуючи його ObjectId).

Параметр timestamps: true додає до схеми два додаткових поля: createdAt і updatedAt, які автоматично відстежують часові позначки створення та модифікації завдання.

Маршрут **projects/{projectId}/taskCreate** відповідає за створення нового завдання в рамках конкретного проекту. Він очікує, що ідентифікатор проекту буде включено як параметр до URL-адреси ({projectId}), що дозволить серверу ідентифікувати проект, з яким буде пов'язано завдання.

```
models > .js task.js > ...
1  const mongoose = require('mongoose');
2
3  const taskSchema = new mongoose.Schema({
4    title: {
5      type: String,
6      required: true,
7    },
8    description: {
9      type: String,
10     required: true,
11   },
12   assignedTo: {
13     type: String,
14     required: true,
15   },
16   dueDate: {
17     type: Date,
18     required: true,
19   },
20   status: {
21     type: String,
22     enum: ['To Do', 'In Progress', 'Completed'],
23     default: 'To Do',
24   },
25   project: {
26     type: mongoose.Schema.Types.ObjectId,
27     ref: 'Project',
28     required: true,
29   },
30 }, {
31   timestamps: true,
32 });
33
34 const Task = mongoose.model('Task', taskSchema);
35
36 module.exports = Task;
```

Рисунок. 3.10 – Модель Task.js для зберігання сутностей задач

Коли клієнт робить запит POST до цього маршруту, сервер обробить запит за допомогою відповідного обробника маршруту. Реалізація наведена на рис. 3.11

У цій реалізації обробник маршруту спочатку витягує ідентифікатор проекту з параметра URL (:projectId) і необхідні дані завдання з тіла запиту (назва, опис, assignedTo, dueDate).

```
controllers > JS task-controller.js > ...
11
12 app.post('/projects/:projectId/taskCreate', async (req, res) => {
13   try {
14     const { projectId } = req.params;
15     const { title, description, assignedTo, dueDate } = req.body;
16
17     const project = await Project.findById(projectId);
18
19     if (!project) {
20       return res.status(404).json({ message: 'Project not found' });
21     }
22
23     const task = new Task({
24       title,
25       description,
26       assignedTo,
27       dueDate,
28       project: projectId,
29     });
30
31     const savedTask = await task.save();
32
33     project.tasks.push(savedTask._id);
34     await project.save();
35
36     return res.status(201).json({ message: 'Task created successfully', task: savedTask });
37   } catch (error) {
38     return res.status(400).json({ message: 'Error creating task', error: error.message });
39   }
40 });
```

Рисунок 3.11 – Лістинг роуту «projects/{projectId}/taskCreate»

Далі він намагається знайти проект за допомогою наданого ідентифікатора проекту за допомогою методу findById() із моделі Project. Якщо проект не існує, він повертає відповідь 404 із відповідним повідомленням про помилку. Якщо проект знайдено, на основі отриманих даних завдання створюється новий об'єкт Task. Властивість проекту встановлюється на ідентифікатор проекту, щоб встановити зв'язок між завданням і проектом. Потім нове завдання зберігається в базі даних за допомогою методу save(), а збережений об'єкт завдання повертається у відповідь. Нарешті ідентифікатор створеного завдання додається до масиву завдань проекту, а проект

					ІАЛЦ.467800.003 ПЗ	Арк.
						60
Зм.	Арк.	№ докум.	Підпис	Дата		

зберігається, щоб забезпечити оновлення зв'язку. Якщо під час процесу виникають будь-які помилки, наприклад помилки перевірки чи помилки бази даних, клієнту повертається відповідна відповідь про помилку.

Коли клієнт робить запит POST до маршруту **projects/{projectId}/taskUpdate/{taskId}**, сервер обробить запит за допомогою відповідного обробника маршруту. Ось приклад реалізації (рис 3.11): цей маршрут відповідає за оновлення існуючого завдання в рамках конкретного проекту. Він очікує, що ідентифікатор проекту та ідентифікатор завдання будуть включені як параметри в URL-адресу ({projectId} і {taskId} відповідно), дозволяючи серверу ідентифікувати проект і завдання, які потрібно оновити.

У цій реалізації (рис. 3.12) обробник маршруту спочатку витягує ідентифікатор проекту та ідентифікатор завдання з параметрів URL-адреси (:projectId і :taskId відповідно) і необхідні дані завдання з тіла запиту (заголовок, опис, assignedTo, dueDate, статус). Далі він намагається знайти проект за допомогою наданого ідентифікатора проекту за допомогою методу findById() із моделі Project. Якщо проект не існує, він повертає відповідь 404 із відповідним повідомленням про помилку. Подібним чином він намагається знайти завдання за допомогою наданого ідентифікатора завдання за допомогою методу findById() із моделі Task. Якщо завдання не існує, воно повертає відповідь 404 із відповідним повідомленням про помилку.

Якщо знайдено і проект, і завдання, у відповідь повертається об'єкт завдання, що дозволяє клієнту отримати та відобразити деталі завдання. Якщо під час процесу виникають будь-які помилки, наприклад помилки бази даних або інші винятки, клієнту повертається відповідна відповідь про помилку.

Маршрут GET projects/{projectId}/task/{taskId} відповідає за отримання деталей конкретного завдання в рамках проекту. Він очікує, що ідентифікатор проекту та ідентифікатор завдання будуть включені як параметри в URL-адресу ({projectId} і {taskId} відповідно), дозволяючи серверу ідентифікувати проект і завдання, які потрібно отримати.

					ІАЛЦ.467800.003 ПЗ	Арк.
						61
Зм.	Арк.	№ докум.	Підпис	Дата		

Він очікує, що ідентифікатор проекту та ідентифікатор завдання будуть включені як параметри в URL-адресу ({projectId} і {taskId} відповідно), дозволяючи серверу ідентифікувати проект і завдання, які потрібно отримати.

```
45 app.post('/projects/:projectId/taskUpdate/:taskId', async (req, res) => {
46   try {
47     const { projectId, taskId } = req.params;
48     const { title, description, assignedTo, dueDate, status } = req.body;
49
50     const project = await Project.findById(projectId);
51
52     if (!project) {
53       return res.status(404).json({ message: 'Project not found' });
54     }
55     const task = await Task.findById(taskId);
56
57     if (!task) {
58       return res.status(404).json({ message: 'Task not found' });
59     }
60
61     task.title = title;
62     task.description = description;
63     task.assignedTo = assignedTo;
64     task.dueDate = dueDate;
65     task.status = status;
66
67     const updatedTask = await task.save();
68
69     return res.json({ message: 'Task updated successfully', task: updatedTask });
70   } catch (error) {
71     return res.status(400).json({ message: 'Error updating task', error: error.message });
72   }
73 });
```

Рисунок 3.12 – Лістинг роуту «projects/{projectId}/taskUpdate/{taskId}»

Коли клієнт робить запит GET до цього маршруту, сервер оброблятиме запит за допомогою відповідного обробника маршруту. Приклад реалізації на рис 3.13. У цій реалізації обробник маршруту спочатку витягує ідентифікатор проекту та ідентифікатор завдання з параметрів URL (:projectId і :taskId відповідно). Далі він намагається знайти проект за допомогою наданого ідентифікатора проекту за допомогою методу findById() із моделі Project. Якщо проект не існує, він повертає відповідь 404 із відповідним повідомленням про помилку.

Подібним чином він намагається знайти завдання за допомогою наданого ідентифікатора завдання за допомогою методу `findById()` із моделі `Task`.

```
app.get('/projects/:projectId/task/:taskId', async (req, res) => {
  try {
    const { projectId, taskId } = req.params;

    const project = await Project.findById(projectId);

    if (!project) {
      return res.status(404).json({ message: 'Project not found' });
    }

    const task = await Task.findById(taskId);

    if (!task) {
      return res.status(404).json({ message: 'Task not found' });
    }

    return res.json({ task });
  } catch (error) {
    return res.status(400).json({ message: 'Error fetching task', error: error.message });
  }
});
```

Рисунок 3.13 – Лістинг роуту «projects/{projectId}/task/{taskId}»

Якщо завдання не існує, воно повертає відповідь 404 із відповідним повідомленням про помилку. Якщо знайдено і проект, і завдання, у відповідь повертається об’єкт завдання, що дозволяє клієнту отримати та відобразити деталі завдання. Якщо під час процесу виникають будь-які помилки, наприклад помилки бази даних або інші винятки, клієнту повертається відповідна відповідь про помилку.

3.4.4 Методи для керування коментарями

Модель “Comment” визначається за допомогою класу `mongoose.Schema`, який надає спосіб визначення структури та правил перевірки даних.

Схема «Comment» містить такі поля:

- **text** - представляє текстовий вміст коментаря. Він має тип `String` і є обов’язковим
- **author** - представляє автора коментаря. Це посилання на модель «Користувач», що пов’язує коментар із користувачем, який його створив. Це поле є обов’язковим.

- **createdAt** - представляє мітку часу, коли було створено коментар. Він має тип Дата і має значення за замовчуванням (поточну дату й час)
- **project** - представляє проект, до якого належить коментар. Це посилання на модель "Проект", яке є обов'язковим.
- **task** - представляє завдання, з яким пов'язаний коментар. Це посилання на модель "Завдання" яке є обов'язковим.

Потім функція `mongoose.model()` використовується для створення моделі «Comment» на основі визначеної схеми. Реалізація моделі вказана на рис. 3.14

Оскільки методи для роботи з коментарями принципово не відрізняються від інших сутностей, розглянемо лише команду для створення нового коментаря під задачею.

Маршрут `POST projects/{projectId}/task/{taskId}/createComment` у веб-програмі керування розробкою проекту за допомогою Express відповідає за обробку створення нового коментаря до певного завдання в рамках проекту.

Цей маршрут дозволяє клієнту надсилати запит `POST` з ідентифікатором проекту та ідентифікатором завдання як параметрами в URL-адресі разом із даними коментаря в тілі запиту.

```
models > JS comment.js > ...
1  const mongoose = require('mongoose');
2
3  const commentSchema = new mongoose.Schema({
4    text: {
5      type: String,
6      required: true,
7    },
8    author: {
9      type: mongoose.Schema.Types.ObjectId,
10     ref: 'User',
11     required: true,
12   },
13   createdAt: {
14     type: Date,
15     default: Date.now,
16   },
17   project: {
18     type: mongoose.Schema.Types.ObjectId,
19     ref: 'Project',
20     required: true,
21   },
22   task: {
23     type: mongoose.Schema.Types.ObjectId,
24     ref: 'Task',
25     required: true,
26   },
27 });
28
29 const Comment = mongoose.model('Comment', commentSchema);
30
31 module.exports = Comment;
```

Рисунок 3.14 – Модель `Comment.js` для зберігання сутностей користувачів

З огляду на реалізацію (рис 3.15) метод `Express app.post()` використовується для визначення обробника запиту `POST` для вказаного

маршруту. Заповнювачі: `projectId` і `taskId` в URL-адресі маршруту є динамічними параметрами, доступ до яких можна отримати за допомогою об'єкта `req.params`.

Всередині обробника маршруту об'єкт `req.body` деструктуровано, щоб витягти автора та текстові поля даних коментаря. Ви можете виконати будь-яку необхідну перевірку даних, щоб переконатися, що вони відповідають необхідним критеріям.

Далі створюється новий об'єкт коментаря з витягнутими даними разом із додатковою інформацією, такою як `projectId`, `taskId` і мітка часу. Ви можете налаштувати структуру та властивості об'єкта коментаря відповідно до вимог програми.

```
app.post('/projects/:projectId/task/:taskId/createComment', async (req, res) => {
  const projectId = req.params.projectId;
  const taskId = req.params.taskId;
  const { author, text } = req.body;

  if (!author || !text) {
    return res.status(400).json({ message: 'Author and text are required' });
  }

  try {
    const comment = new Comment({
      projectId,
      taskId,
      author,
      text,
      timestamp: Date.now()
    });

    await comment.save();

    res.json({ message: 'Comment created successfully', comment });
  } catch (error) {
    res.status(500).json({ message: 'Failed to create comment', error });
  }
});
```

Рисунок 3.15 – Лістинг роуту

«`projects/{projectId}/task/{taskId}/createComment`»

Після створення об'єкта коментаря ви можете реалізувати свою логіку, щоб зберегти коментар до бази даних або будь-якого іншого джерела даних. Це може включати використання ORM (наприклад, Mongoose) або пряме виконання запитів до бази даних.

Нарешті клієнту повертається відповідь у форматі JSON. Яка вказує на успішне створення коментаря та включає створений об'єкт коментаря в корисне навантаження відповіді.

3.5 Розробка клієнтської частини

3.5.1 Огляд

Клієнтська частина додатку була створена за допомогою CRA (create-react-app). Create React App — це популярний інструмент командного рядка, який дозволяє розробникам швидко налаштувати та створювати новий проект React із попередньо визначеною конфігурацією. Він забезпечує спрощений та ефективний спосіб розпочати створення додатків React без необхідності складного налаштування та налаштування.

В порівнянні з іншим інструментами для створення клієнтських застосунків, він має декілька переваг:

1. Create React App позбавляє від необхідності налаштовувати процес збірки, веб-пакет та інші залежності вручну. За допомогою однієї команди ви можете створити новий проект React із попередньо налаштованим середовищем розробки, включаючи сервер розробки, гаряче перезавантаження модулів і оптимізовані робочі збірки.
2. Додаток Create React поставляється з попередньо визначеним набором параметрів конфігурації та інструментів, що дозволяє розробникам легко розпочати кодування, не витрачаючи час на початкове налаштування. Однак він також забезпечує гнучкість налаштування та заміни конфігурації за замовчуванням, якщо це необхідно.
3. Додаток Create React включає вбудований сервер розробки, який дозволяє змінювати код у реальному часі та перезавантажувати його. Це дозволяє розробникам бачити негайні наслідки змін коду, не оновлюючи браузер вручну.

					ІАЛЦ.467800.003 ПЗ	Арк.
						66
Зм.	Арк.	№ докум.	Підпис	Дата		

4. Додаток Create React обробляє конфігурацію Babel і Webpack, які є основними інструментами для транспіляції та об'єднання коду JavaScript. Він автоматично налаштовує Babel для підтримки сучасного синтаксису JavaScript і транспілює його у сумісну версію для різних браузерів.
5. Create React App надає набір вбудованих сценаріїв, які дозволяють легко виконувати звичайні завдання розробки. Ці сценарії включають запуск сервера розробки, створення оптимізованих робочих збірок, виконання тестів тощо. Це позбавляє від необхідності налаштовувати та керувати сценаріями вручну.
6. Create React App використовує Babel та інші плагіни для забезпечення підтримки сучасних функцій JavaScript, таких як функції зі стрілками, шаблонні літерали, деструктурування та синтаксис async/await. Це дозволяє розробникам писати чистіший і лаконічніший код, не турбуючись про сумісність браузера.
7. Хоча Create React App постачається з попередньо визначеним набором конфігурацій і залежностей, він також дозволяє розробникам розширювати та налаштовувати проект за потреби. Ви можете вилучити конфігурацію проекту, щоб мати повний контроль над процесом побудови та залежностями, уможлививлюючи розширене налаштування та інтеграцію з додатковими інструментами.

Загалом, Create React App забезпечує зручний та ефективний спосіб розпочати створення програм React, абстрагуючись від початкових складнощів налаштування та конфігурації. Це дозволяє розробникам зосередитися на написанні коду та створенні інтерфейсів користувача, не турбуючись про інструменти та конфігурацію збірки. Незалежно від того, початківець ви чи досвідчений розробник, Create React App пропонує міцну основу для розробки додатків React у безпроблемний спосіб.

					ІАЛЦ.467800.003 ПЗ	Арк.
						67
Зм.	Арк.	№ докум.	Підпис	Дата		

3.5.2 Реалізація сторінки з авторизацією

Код на рис. 3.16 є компонентом React під назвою LoginForm, який представляє форму входу. Він використовує різні бібліотеки та компоненти для роботи з відтворенням форм і керування станом. Компонент імпортує необхідні стилі та значки із зовнішніх файлів. Він також імпортує хук useState з бібліотеки React і компоненти Formik (Field, Form) з бібліотеки formik. Крім того, він імпортує компоненти з бібліотеки Material-UI, такі як TextField і Button. Material-UI надає широкий спектр готових до використання компонентів, таких як TextField, Button, FormControl тощо. Ці компоненти легко налаштовуються та можуть заощадити ваш час і зусилля при створенні елементів інтерфейсу користувача для вашої сторінки авторизації.

```
client > src > login > login-form > JS login-form.js > ...
1 > import './login-form.css'; ...
9
10 function LoginForm() {
11   const [authorizationType, setAuthorizationType] = useState();
12
13   const authorizationTypeHandler = (type) => {
14     setAuthorizationType(type);
15   };
16
17 > if (authorizationType === 'login') { ...
55 }
56
57 > if (authorizationType === 'register') { ...
78 }
79
80 return (
81   <div class="loginWrapper">
82     <div class="header">Sprintman</div>
83
84     <div class="login-block-wrapper">
85       <div class="choice-block" onClick={() => authorizationTypeHandler('login')}>
86         <img src={login} className="choice-logo" alt="login" />
87         <div class="choice-text">sign up</div>
88       </div>
89       <div class="choice-block" onClick={() => authorizationTypeHandler('register')}>
90         <img src={register} className="choice-logo" alt="login" />
91         <div class="choice-text">create account</div>
92       </div>
93     </div>
94   </div>
95 );
96 }
97
98 export default LoginForm;
```

Рисунок 3.16 – Лістинг компоненту LoginForm.js

Компонент визначає змінну стану authorizationType за допомогою хука useState. Ця змінна визначає, який розділ форми входу відобразити на основі вибраного типу авторизації. Компонент містить умовне відтворення за допомогою операторів if. Якщо для атрибута authorizationType встановлено значення «login», форма входу відображається з електронною поштою та паролем, а також кнопкою входу. Значеннями вхідних даних керуються за

					ІАЛЦ.467800.003 ПЗ	Арк.
						68
Зм.	Арк.	№ докум.	Підпис	Дата		

допомогою змінних стану, а надсилання форми наразі не обробляється (`onSubmit={() => {}}`). Якщо для атрибута `authorizationType` встановлено значення «register», форма реєстрації відображається за допомогою компонента `Formik`. Форма збирає ім'я та електронну адресу користувача, а після надсилання сповіщає про значення форми. Якщо жодна з наведених вище умов не виконується, відображається розділ із параметрами входу та реєстрації. Натискання на параметри запускає функцію `authorizationTypeHandler`, яка оновлює стан `authorizationType` на основі вибраного параметра.

Компонент використовує класи `CSS` і вбудовані стилі для визначення макета та стилю елементів форми входу (рис 3.17 і 3.20).

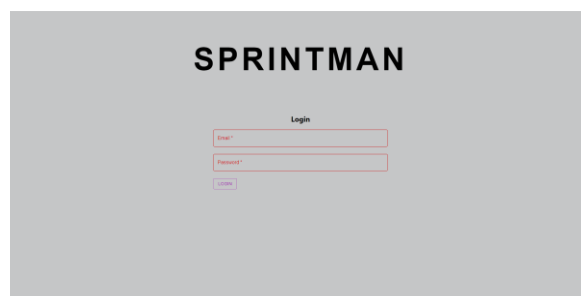
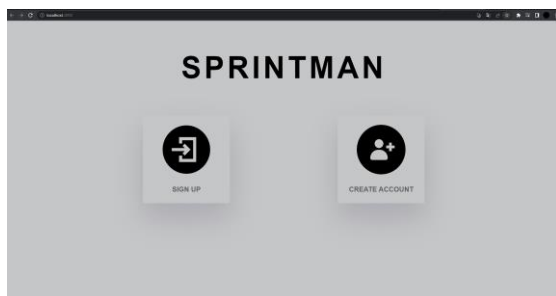


Рисунок 3.17, 3.18 – Приклад сторінки з авторизацією користувача

3.5.3 Огляд бібліотеки `MIU`

`Material-UI` — популярна бібліотека інтерфейсу користувача з відкритим вихідним кодом для `React`, яка дозволяє розробникам створювати красиві та адаптивні інтерфейси користувача. Він базується на розроблених `Google` інструкціях з матеріального дизайну, які містять набір принципів і шаблонів дизайну для створення сучасних і візуально привабливих веб-додатків. `Material-UI` надає повну колекцію готових до використання компонентів, таких як кнопки, форми, діалоги, картки та багато іншого. Ці компоненти розроблено таким чином, щоб їх можна було налаштувати та багаторазово використовувати, що дозволяє розробникам створювати послідовні та візуально приємні елементи інтерфейсу користувача у своїх програмах. `Material-UI` дотримується принципів адаптивного дизайну, гарантуючи, що компоненти виглядатимуть і добре поведуться на різних розмірах екрана та на різних

пристроях. Він забезпечує адаптивні контрольні точки та службові класи CSS, щоб легко адаптувати макет і стиль на основі пристрою чи розміру вікна перегляду. Material-UI створено з урахуванням доступності. Він містить вбудовані функції доступності, такі як правильна семантична розмітка, підтримка навігації з клавіатури та атрибути ARIA, щоб забезпечити доступність компонентів для користувачів з обмеженими можливостями. Це допомагає створювати інклюзивні та зручні програми. Material-UI добре інтегрується з іншими популярними бібліотеками та інструментами в екосистемі React. Він підтримує інтеграцію з бібліотеками керування станом, такими як Redux і MobX, бібліотеками форм, такими як Formik і React Hook Form, і бібліотеками маршрутизації, такими як React Router. Це дозволяє розробникам використовувати переваги Material-UI, одночасно легко інтегруючись із наявним стеком. Material-UI має велике та активне співтовариство розробників, що означає, що існує багато ресурсів, навчальних посібників і прикладів, які допоможуть розробникам почати роботу та вирішити проблеми. Бібліотека добре підтримується та регулярно оновлюється новими функціями та виправленнями помилок, що забезпечує її надійність і стабільність.

Загалом Material-UI пропонує поєднання візуальної привабливості, швидкості реагування, параметрів налаштування та відповідності доступності, що робить його відповідним вибором для веб-додатку для керування розробкою проектів. Він забезпечує надійну основу для побудови інтуїтивно зрозумілого та візуально привабливого інтерфейсу, заощаджуючи час на розробку та забезпечуючи узгоджену роботу користувачів на різних пристроях.

Як приклад використання MIU можна розглянути компонент ProjectForm, який відображає форму для створення проекту. Хук makeStyles від Material-UI використовується для визначення власних стилів для компонентів у формі. Це дозволяє нам легко налаштувати зовнішній вигляд і компонування елементів форми відповідно до вимог дизайну нашої програми.

					ІАЛЦ.467800.003 ПЗ	Арк.
						70
Зм.	Арк.	№ докум.	Підпис	Дата		

Компонент `TextField` використовується для візуалізації полів введення для збору інформації, пов'язаної з проектом, як-от назва проекту, опис, дата початку та дата завершення. Ці поля налаштовані за допомогою таких атрибутів, як `label`, `variant`, `fullWidth` і `type`, щоб забезпечити відповідні мітки, стиль і типи введення.

Компонент `Grid` використовується для створення адаптивного макета форми. Він використовує властивості контейнера та елемента для створення структури на основі сітки з рядками та стовпцями. Це допомагає досягти узгодженого та організованого макета для полів форми.

Компонент `Button` використовується для надання кнопок дій у формі. У цьому випадку ми маємо кнопку «Створити проект» для надсилання форми та кнопку «Скасувати» для скасування надсилання форми. Кнопки оформлені за допомогою властивостей варіантів і кольорів відповідно до вказівок щодо дизайну `Material-UI`.

Приклад реалізації можна бачити на рис. 3.19

```
const useStyles = makeStyles((theme) => ({
  root: {
    flexGrow: 1,
    padding: theme.spacing(3),
  },
  paper: {
    padding: theme.spacing(2),
  },
  textField: {
    marginBottom: theme.spacing(2),
  },
  button: {
    marginRight: theme.spacing(2),
  },
}));

function ProjectForm() {
  const classes = useStyles();

  const handleSubmit = (event) => { ...
  };

  return (
    <div className={classes.root}>
      <Paper className={classes.paper}>
```

Рисунок 3.19— Приклад використання `Material UI` в компоненті `ProjectForm`

3.5.4 Використання користувацьких хуків

Користувальницькі хуки в `React` — це потужна функція, яка дозволяє витягувати багаторазову логіку з компонентів. Вони дають змогу створювати

					ІАЛЦ.467800.003 ПЗ	Арк.
						71
Зм.	Арк.	№ докум.	Підпис	Дата		

функції, які можуть містити логіку стану, побічні ефекти або будь-які інші функції, які ви хочете повторно використовувати в кількох компонентах.

Щоб використовувати спеціальний хук, ви просто імпортуєте та викликаєте його у функціональному компоненті. Користувальницькі хуки дотримуються умов іменування, починаючи зі слова «використовувати», щоб було зрозуміло, що це хуки.

Основна перевага використання користувальницьких хуків — можливість повторного використання коду. Витягнувши певну логіку в спеціальний хук, ви можете легко повторно використовувати цю логіку в різних компонентах без дублювання коду. Це сприяє більш чистому, більш модульному коду та зменшує кількість шаблонів, які вам потрібно написати.

Користувацькі хуки можуть інкапсулювати широкий спектр функцій, таких як керування станом, обробка підписок, отримання даних з API, обробка перевірки форми тощо. Вони також можуть забезпечити чистий інтерфейс для зв'язку компонентів або інкапсулювати складну логіку, якою може бути важко керувати в рамках одного компонента.

Користувальницькі хуки можуть отримати доступ до інших хуків React, таких як `useState`, `useEffect` або `useContext`, дозволяючи вам використовувати існуючий API хуків у ваших користувацьких хуках. Це означає, що ви можете створювати хуки, які надають додаткову функціональність поверх існуючих хуків, розширюючи можливості ваших компонентів.

Створюючи нестандартні хуки, важливо дотримуватися правил хуків. Це означає, що користувацькі хуки повинні викликати лише інші хуки або інші користувацькі хуки. Їх не можна назвати умовно або всередині циклів.

Використання користувацьких хуків може значно покращити можливість повторного використання та обслуговування вашої кодової бази. Вони дозволяють витягувати та ділитися спільною логікою між компонентами, заохочуючи модульний та ефективний підхід до розробки. Використовуючи

					ІАЛЦ.467800.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		72

користувачки хуки, ви можете створювати більш масштабовані та зручні в обслуговуванні програми React.

Як приклад використання користувачки хуків можна розглянути хук `useAuth` (рис. 3.20)

```
client > src > hooks > JS auth.hook.js > useAuth
1  import { useState, useCallback, useEffect } from 'react';
2
3  const storageName = 'userData';
4
5  export const useAuth = () => {
6    const [token, setToken] = useState(null);
7    const [ready, setReady] = useState(false);
8    const [userId, setUserId] = useState(null);
9    const [email, setEmail] = useState(null);
10
11   const login = useCallback((jwtToken, id, email) => {
12     setToken(jwtToken);
13     setUserId(id);
14     setEmail(email);
15
16     localStorage.setItem(
17       storageName,
18       JSON.stringify({
19         userId: id,
20         token: jwtToken,
21         email,
22       })
23     );
24   }, []);
25
26   const logout = useCallback(() => { ...
32   }, []);
33
34   useEffect(() => {
35     const data = JSON.parse(localStorage.getItem(storageName));
36
37     if (data && data.token) {
38       login(data.token, data.userId, data.currentEmail);
39     }
40     setReady(true);
41   }, [login]);
42
43   return { login, logout, token, userId, ready, email };
44 };
45
```

Рисунок 3.20– Приклад використання користувачки хуків на прикладі хука `useAuth`

Хук використовує хуки `useState`, `useCallback` і `useEffect` від React. Він визначає кілька змінних стану, зокрема `token`, `ready`, `userId` та `email`, які використовуються для зберігання даних, пов'язаних з автентифікацією.

Функція входу — це зворотній виклик, який приймає токен JWT, ідентифікатор користувача та електронну адресу як параметри. Він оновлює

змінні стану token, userId і email наданими значеннями та зберігає їх у локальному сховищі браузера. Це дозволяє користувачеві залишатися в системі, навіть якщо він оновлює сторінку або повторно відкриває програму.

Функція виходу з системи — це інший зворотній виклик, який очищає змінні стану та видаляє дані автентифікації з localStorage.

Хук useEffect використовується для отримання даних автентифікації з localStorage під час монтування компонента. Якщо в localStorage є дійсний маркер, функція входу викликається зі збереженим маркером, ідентифікатором користувача та електронною поштою, щоб встановити початковий стан автентифікації. Для змінної стану готовності встановлено значення true, щоб вказати, що налаштування автентифікації завершено.

Хук useAuth повертає об'єкт зі значеннями login, logout, token, userId, ready та email. Ці значення можна використовувати в інших компонентах для керування потоком автентифікації та доступу до інформації автентифікованого користувача.

Загалом хук useAuth забезпечує зручний спосіб обробки автентифікації в програмі React, дозволяючи компонентам легко отримувати доступ і оновлювати стан автентифікації.

ВИСНОВОК ДО РОЗДІЛУ 3

Третій розділ дав вичерпний огляд інструментів і методів розробки, які використовуються у веб-додатку для керування розробкою проекту. Вибір React як бібліотеки інтерфейсної розробки був виправданий на основі її компонентної архітектури, багаторазового використання та розгалуженої екосистеми бібліотек та інструментів. React дозволяє створювати динамічні та чуйні користувацькі інтерфейси, сприяючи розширеному досвіду користувача.

Використання Express як базової системи розробки також було виправданим. Express — це швидка та мінімалістична структура веб-додатків для Node.js, яка забезпечує надійні можливості маршрутизації та модульну структуру для створення серверної логіки. Він забезпечує ефективну обробку HTTP-запитів і відповідей, що робить його придатним для створення масштабованих і продуктивних API.

Описано впровадження JWT (веб-токенів JSON) для автентифікації та авторизації. JWT надає безпечний механізм автентифікації користувача без збереження стану шляхом створення та перевірки маркерів. Це гарантує, що лише автентифіковані користувачі можуть отримати доступ до захищених маршрутів і ресурсів, підвищуючи безпеку програми.

Реалізовано кінцеві точки для створення користувачів, що дозволяє реєструвати нових користувачів у системі. Ці кінцеві точки спрощують зберігання даних користувача в базі даних і забезпечують безпечну обробку паролів за допомогою таких методів, як хешування та шифрування.

Крім того, реалізовано роути для створення та оновлення проектів. Вони дозволяють керувати інформацією, пов'язаною з проектом, такою як деталі проекту, завдання та терміни. За допомогою цих ендпоінтів користувачі можуть створювати нові проекти, оновлювати існуючі проекти та відстежувати їх прогрес протягом життєвого циклу розробки.

					ІАЛЦ.467800.003 ПЗ	Арк.
						75
Зм.	Арк.	№ докум.	Підпис	Дата		

Було описано використання Material-UI (MIU), демонструючи його переваги у створенні візуально привабливого та узгодженого інтерфейсу користувача. MIU надає широкий спектр готових компонентів і стилів, які можна легко налаштувати відповідно до вимог проекту. Він покращує процес розробки, пропонуючи адаптивну та доступну структуру інтерфейсу користувача.

Нарешті, було оглянуто використання власних хуків. Спеціальні хуки відповідають інкапсуляцію та повторне використання логіки в кількох компонентах, сприяючи модульності коду та зменшуючи дублювання коду. Наведений приклад демонструє використання спеціального підхоплення для обробки автентифікації, керування маркером, інформацією про користувача та станом сеансу.

Загалом поєднання React, Express, JWT, Material-UI і користувацьких хуків призвело до розробки надійного веб-додатку для керування проектами. Ці інструменти та методи покращили взаємодію з користувачем, безпеку та зручність обслуговування програми, завдяки чому вона добре підходить для ефективного керування та відстеження проектів.

					ІАЛЦ.467800.003 ПЗ	Арк.
						76
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

Під час виконання дипломної роботи було забезпечене комплексне дослідження процесу створення веб-сторінок і побудови веб-серверів, кульмінацією якого стала реалізація веб-додатку для управління розробкою проекту.

У першому розділі розглядалися різні методи та технології створення веб-сторінок, висвітлюючи їхні сильні та слабкі сторони та випадки використання. Він досліджував традиційні підходи, такі як HTML, CSS і JavaScript, а також сучасні фреймворки та бібліотеки, такі як React, Vue і Angular. Розуміючи різні доступні варіанти, були отримані знання про вибір найбільш підходящих інструментів для своїх проектів веб-розробки.

У другому розділі розглядалися принципи побудови веб-серверів, зосереджуючись на серверній архітектурі та функціональності. У ньому описані фундаментальні концепції зв'язку сервер-клієнт, протоколи HTTP та RESTful API. Крім того, було досліджено такі популярні серверні фреймворки, як Express, і виділено їхні функції для обробки маршрутизації, проміжного програмного забезпечення та інтеграції бази даних. Розуміючи ці принципи, були отримані необхідні знання для створення надійних і масштабованих веб-серверів.

У третьому розділі було представлено практичну реалізацію веб-додатку для управління розробкою проекту. Він продемонстрував використання React для розробки на стороні клієнта, забезпечуючи динамічний та інтерактивний інтерфейс користувача. Інтеграція Express на стороні сервера сприяла безперебійному спілкуванню з клієнтом і керувала бізнес-логікою та збереженням даних. Реалізація також включала такі функції, як автентифікація користувачів, створення проектів, керування завданнями та сповіщення.

Об'єднавши знання, отримані з перших двох розділів, була змога створити комплексну веб-програму, яка відповідала б найкращим практикам і

					ІАЛЦ.467800.003 ПЗ	Арк.
						77
Зм.	Арк.	№ докум.	Підпис	Дата		

галузевим стандартам. Впровадження продемонструвало потужність сучасних інструментів і методів веб-розробки для створення функціональної та зручної програми для керування розробкою проектів.

Загалом ця дипломна робота є цінним ресурсом для розуміння процесу веб-розробки, від створення веб-сторінок до створення веб-серверів. Було продемонстровано практичне застосування цих концепцій шляхом впровадження веб-додатку для управління проектами.

					ІАЛЦ.467800.003 ПЗ	Арк.
						78
Зм.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Alex Banks Learning React: Functional Web Development with React and Redux/ Alex Banks, Eve Porcello – O'Reilly Media, 2017 – С. 3–5.
2. Stoyan Stefanov React: Up & Running: Building Web Applications/ Stoyan Stefanov – O'Reilly Media, 2021 – С. 10-30.
3. Fullstack React: The Complete Guide to ReactJS and Friends/ Anthony Accomazzo, Ari Lerner, and Nate Murray. – Fullstack.io, 2021 – С. 158-159.
4. Node.js Web Development: Server-side Development with Node and Express / David Herron - Packt Publishing, 2020 - С. 89-102.
5. Express in Action: Writing, Building, and Testing Node.js Applications / Evan Hahn - Manning, 2019 - С. 89-102.
6. The Node Beginner Book / Manuel Kiessling- Lulu.com, 2012 - С. 205-206.
7. Learning Node.js: A Hands-On Guide to Building Web Applications in JavaScript / Marc Wandschneider - Addison-Wesley, 2013 - С. 34.
8. MongoDB: The Definitive Guide / Kristina Chodorow, Michael Dirolf- O'Reilly Media, 2013 - С. 76-77.
9. Practical MongoDB: Architecting, Developing, and Administering MongoDB / Shakuntala Gupta Edward, Navin Sabharwal- O'Reilly Media, 2013 - С. 54-55.
- 10.RESTful Web APIs: Services for a Changing World / L. Richardson and M. Amundsen- O'Reilly Media, 2015 - С. 200-202.
- 11.Building Microservices: Designing Fine-Grained Systems / Sam Newman - O'Reilly Media, 2015 - С. 23-26.
- 12.Clean Architecture: A Craftsman's Guide to Software Structure and Design / Robert C. Martin - Pearson, 2017 - С. 400-405.
- 13.Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems / M. Kleppmann - O'Reilly Media, 2017 - С. 45-46.

					ІАЛЦ.467800.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		79

14. Domain-Driven Design: Tackling Complexity in the Heart of Software / Eric Evans - Addison-Wesley Professional, 2003 - C. 154-155.
15. The Pragmatic Programmer: Your Journey to Mastery / A. Hunt, D. Thomas - Pearson, 2002 - C. 69.
16. Agile Software Development, Principles, Patterns, and Practices / Robert C. Martin - Addison-Wesley Professional, 2019 - C. 102.
17. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses / Eric Ries - Currency, 2019 - C. 23-26.
18. User Story Mapping: Discover the Whole Story, Build the Right Product / Jeff Patton - O'Reilly Media, 2014 - C. 141-150.
19. Project Management for the Unofficial Project Manager: A FranklinCovey Title / K. Kogon, S. Blakemore, J. Wood - BenBella Books, 2015 - C. 199.
20. The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win / G. Kim, K. Behr, G. Spafford - IT Revolution Press, 2013 - C. 101

					ІАЛЦ.467800.003 ПЗ	Арк.
						80
Зм.	Арк.	№ докум.	Підпис	Дата		

ДОДАТОК 1

Веб-застосунок для управління розробкою проектів

**Веб застосунок для управління розробкою проектів
(структурна схема)**

ІАЛЩ.467800.004 Д1

Аркушів 1

Київ 2023 р



Браузер

Запит

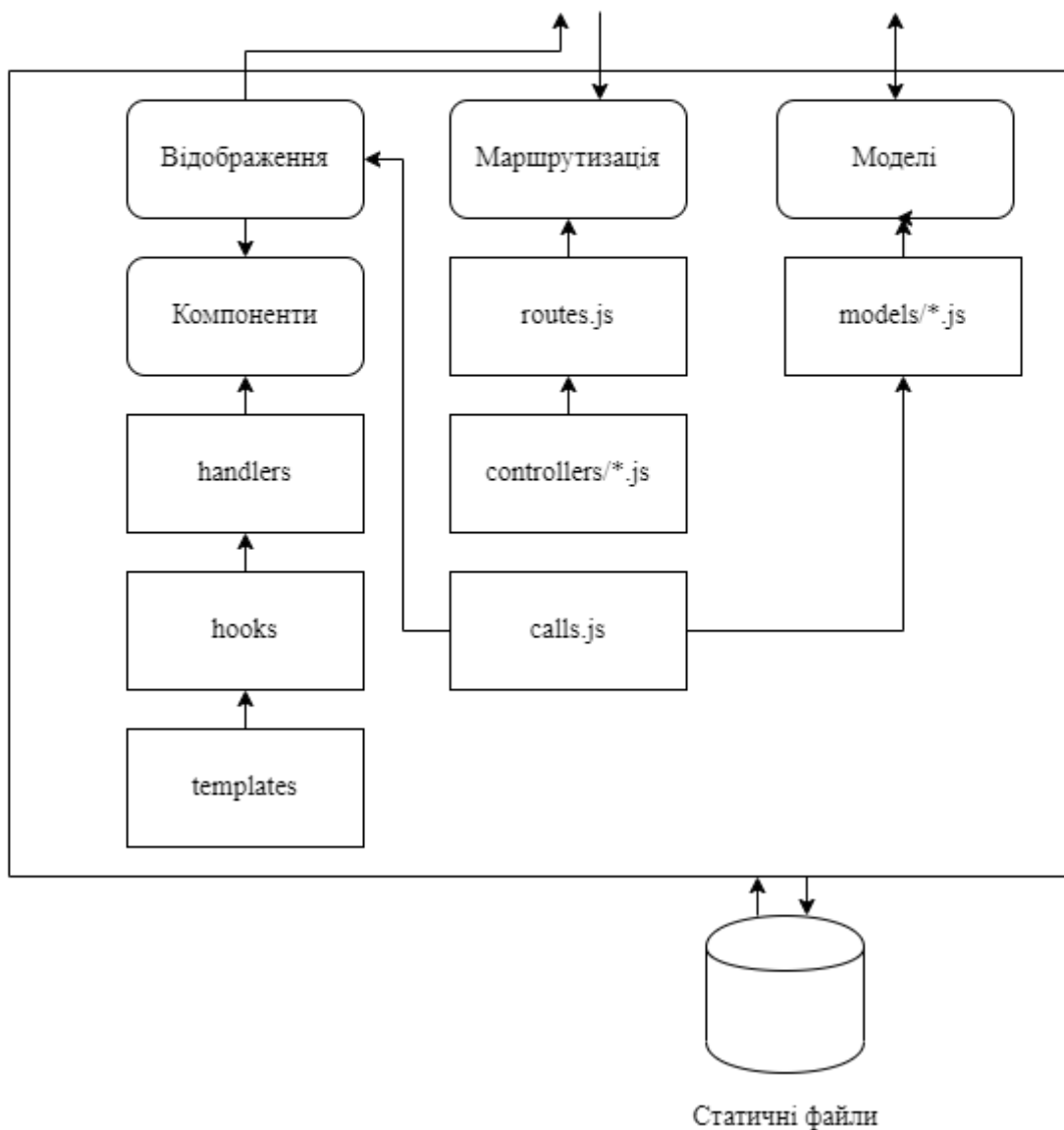
Відповідь



Веб-сервер



База даних



					ІАЛЦ.467800.004 Д1								
		№ докум.	Підпис	Дата									
Розробив		Руденко С.О.			Веб-застосунок для управління розробкою проектів Веб застосунок для управління розробкою проектів (структурна схема)				Літ.	Аркуш	Аркушів		
Перевірив		Русінов В.В.									1	1	
									КПІ ім. Ігоря Сікорського, ФІОТ, ІО-93				
Н. Контр.		Виноградов Ю.М.											
Затвердив													

ДОДАТОК 2

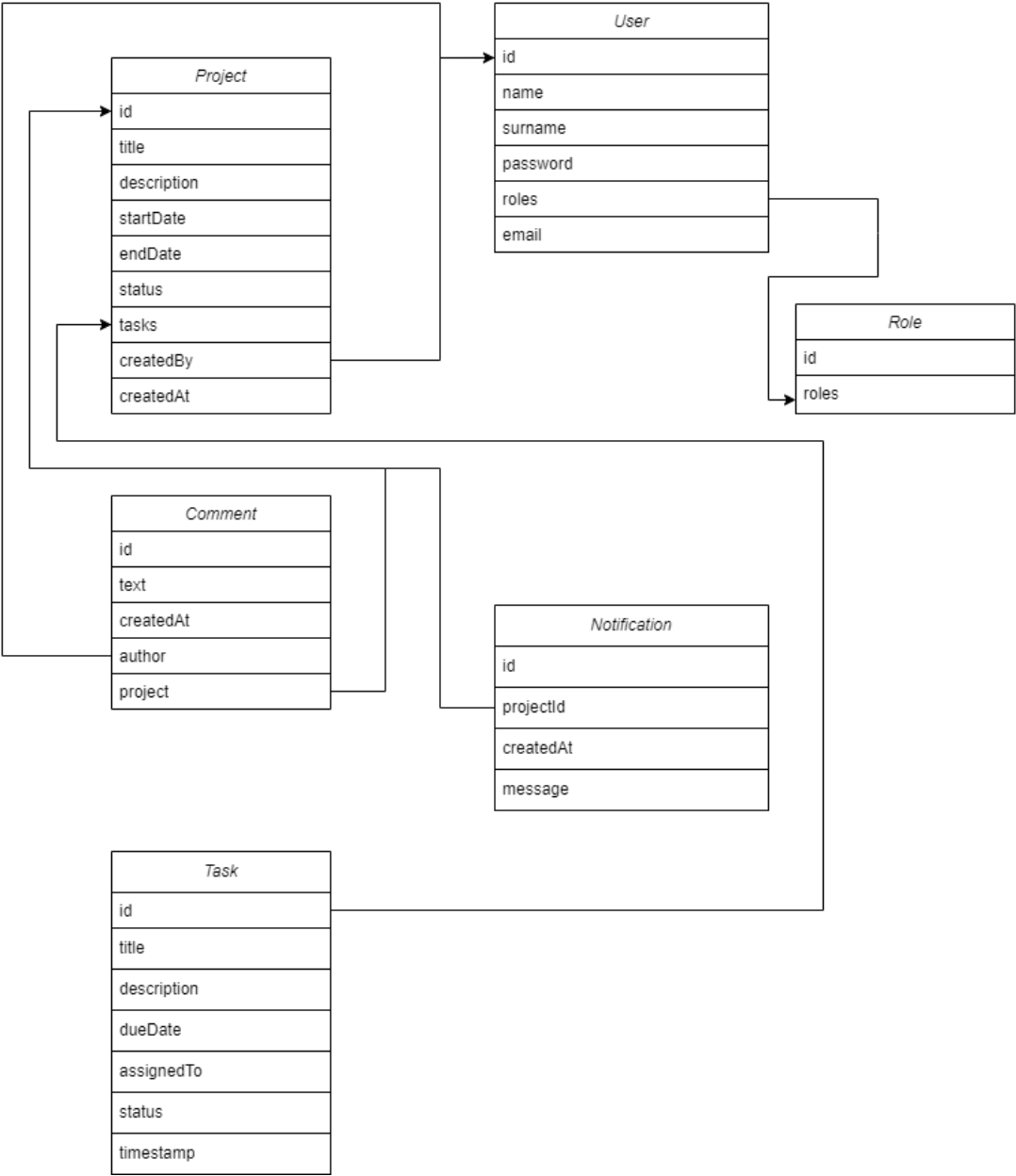
Веб-застосунок для управління розробкою проектів

Веб-застосунок діаграма класів (функціональна схема)

ІАЛЦ.467800.005 Д2

Аркушів 1

Київ 2023 р



					ІАЛЦ.467800.005 Д2			
		№ докум.	Підпис	Дата				
Розробив	Руденко С. О.				Веб-застосунок для управління розробкою проектів Веб-застосунок діаграма класів (функціональна схема)		Літ.	Аркуш
Перевірив	Русінов В.В.							Аркушів
Н. Контр.	Виноградов Ю.М.						КПІ ім. Ігоря	
Затвердив							Сікорського, ФІОТ, ІО-93	
							1	1

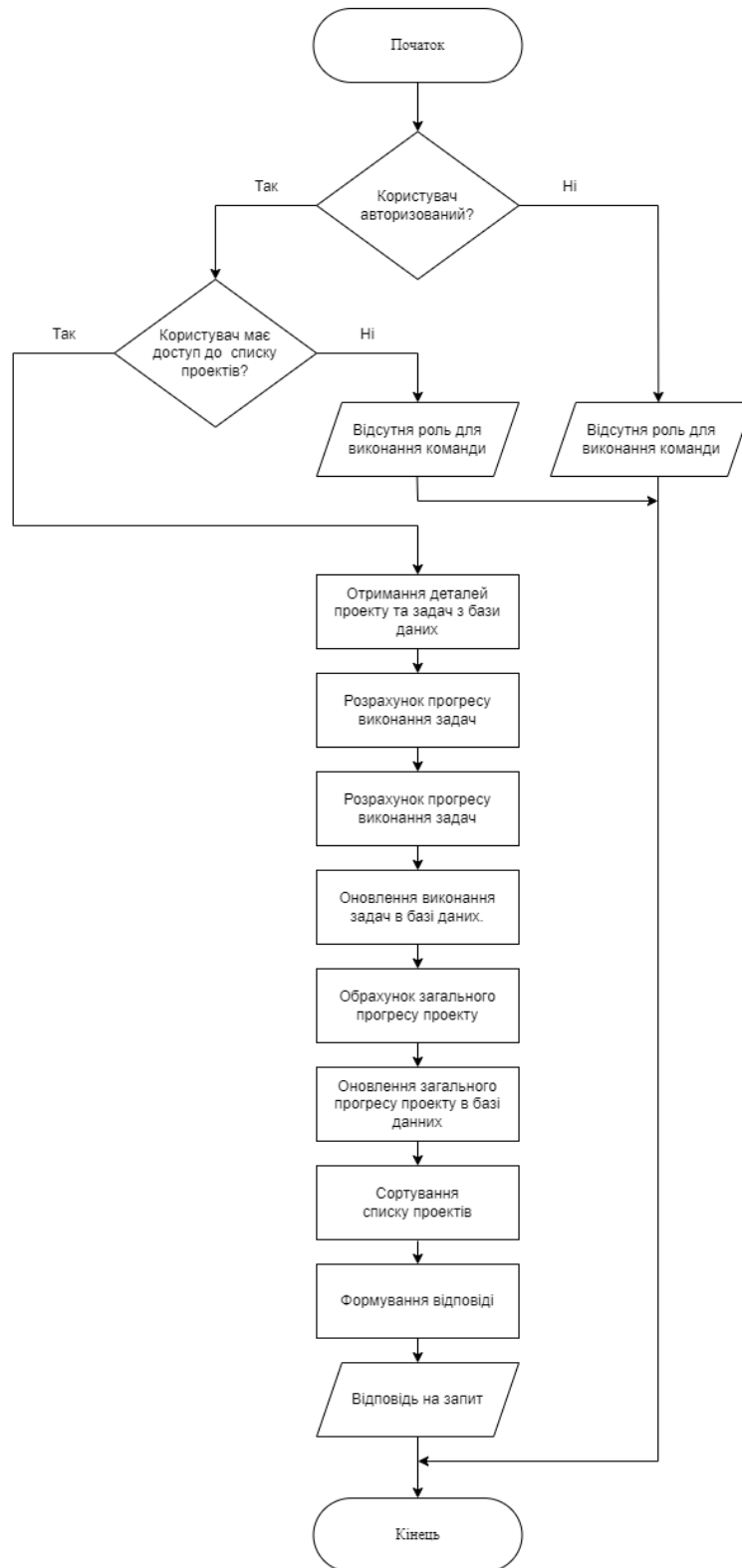
ДОДАТОК 3

Веб-застосунок для управління розробкою проектів

**Алгоритм роботи веб застосунку для управління розробкою
проектів (принципова схема)**
ІАЛЦ.467800.006 ДЗ

Аркушів 1

Київ 2023 р



					ІАЛЦ.467800.006 ДЗ			
		№ докум.	Підпис	Дата	Веб-застосунок для управління розробкою проектів Алгоритм роботи веб застосунку для управління розробкою проектів (принципова схема)	Літ.	Аркуш	Аркушів
Розробив	Руденко С.О.						1	1
Перевірив	Русінов В.В.							
Н. Контр.	Виноградов Ю.М.					КПІ ім. Ігоря Сікорського, ФІОТ, ІО-93		
Затвердив								

ДОДАТОК 4

Веб-застосунок для управління розробкою проектів

Текст програмного коду

ІАЛЦ.467800.007 Д4

Аркушів 24

Київ 2023 р

					ІАЛЦ.467800.007 Д4	Арк.
						9
Зм.	Арк.	№ докум.	Підпис	Дата		

```

{
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "server": "nodemon index.js",
    "client": "npm run start --prefix client",
    "dev": "concurrently \"npm run server\" \"npm run client\""
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "concurrently": "^8.0.1",
    "express": "^4.18.2",
    "express-validator": "^7.0.1",
    "formik": "^2.2.9",
    "jsonwebtoken": "^9.0.0",
    "mongoose": "^7.0.3",
    "nodemon": "^2.0.22"
  }
}
}
}
}

const express = require('express');
const mongoose = require('mongoose');
const authRouter = require('./routes/authorization-router');
const PORT = process.env.PORT || 5000;

const app = express();
app.use(express.json());
app.use('/auth', authRouter);

const start = async () => {
  try {
    await mongoose.connect(`mongodb+srv://rudenkostas2:zqB64AnH3AR8uVMm@kpi-
diploma.oaguzwt.mongodb.net/?retryWrites=true&w=majority`);
    app.listen(PORT, () => console.log('server was started'));
  } catch (e) {
    console.log(e);
  }
};

module.exports = {
  secret: 'SECRET_TOKEN',
};

const Router = require('express');
const router = new Router();
const controller = require('../controllers/authorization-controller');

```

					ІАЛЦ.467800.007 Д4	Арк.
						10
Зм.	Арк.	№ докум.	Підпис	Дата		

```

const { check } = require('express-validator');
const authMiddleware = require('../middleware/authorization-middleware');
const roleMiddleware = require('../middleware/role-middleware');

router.post(
  '/registration',
  [
    check('email', 'Email is required').notEmpty(),
    check('name', 'Name is required').notEmpty(),
    check('surname', 'Name is required').notEmpty(),
    check('password', 'Password is required (length between 4 to 10 characters)').isLength({ min:
4, max: 10 }),
  ],
  controller.registration
);
router.post('/login', controller.login);
router.get('/users', roleMiddleware(['dev']), controller.getUsers);

module.exports = router;

const { Schema, model } = require('mongoose');

const User = new Schema({
  email: { type: String, unique: true, require: true },
  name: { type: String, require: true },
  surname: { type: String, require: true },
  password: { type: String, require: true },
  roles: [{ type: String, ref: 'Role' }],
});

module.exports = model('User', User);
const mongoose = require('mongoose');

const taskSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
  assignedTo: {
    type: String,
    required: true,
  },
},

```

					ІАЛЦ.467800.007 Д4	Арк.
						11
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    dueDate: {
      type: Date,
      required: true,
    },
    status: {
      type: String,
      enum: ['To Do', 'In Progress', 'Completed'],
      default: 'To Do',
    },
    project: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Project',
      required: true,
    },
  }, {
    timestamps: true,
  });

const Task = mongoose.model('Task', taskSchema);

module.exports = Task;

const { Schema, model } = require('mongoose');

const Role = new Schema({
  role: { type: String, require: true, unique: true, default: 'dev' },
});

module.exports = model('Role', Role);

const mongoose = require('mongoose');

const projectSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true
  },
  description: {
    type: String,
    required: true
  },
  startDate: {
    type: Date,
    required: true
  },
  endDate: {
    type: Date,
    required: true
  },
});

```

					ІАЛЦ.467800.007 Д4	Арк.
						12
Зм.	Арк.	№ докум.	Підпис	Дата		

```

status: {
  type: String,
  enum: ['ongoing', 'completed', 'paused'],
  default: 'ongoing'
},
tasks: [
  {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Task'
  }
],
createdBy: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User',
  required: true
},
createdAt: {
  type: Date,
  default: Date.now
}
});

const Project = mongoose.model('Project', projectSchema);

module.exports = Project;

```

```

const mongoose = require('mongoose');

const commentSchema = new mongoose.Schema({
  text: {
    type: String,
    required: true,
  },
  author: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  createdAt: {
    type: Date,
    default: Date.now,
  },
  project: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Project',
    required: true,
  },
});
const Comment = mongoose.model('Comment', commentSchema)
module.exports = Comment;

```

```

const jwt = require('jsonwebtoken');
const { secret } = require('../config');

module.exports = function (req, res, next) {
  if (req.method === 'OPTIONS') {
    next();
  }

  try {
    const token = req.headers.authorization.split(' ')[1];
    if (!token) {
      return res.status(403).json({ message: 'User is not authorized' });
    }
    const decoded = jwt.verify(token, secret);
    req.user = decoded;
    next();
  } catch (e) {
    console.log(e);
    res.status(403).json({ message: 'User is not authorized' });
  }
};

const jwt = require('jsonwebtoken');
const { secret } = require('../config');

module.exports = function (roles) {
  return function (req, res, next) {
    if (req.method === 'OPTIONS') {
      next();
    }

    try {
      const token = req.headers.authorization.split(' ')[1];
      if (!token) {
        return res.status(403).json({ message: 'User is not authorized' });
      }
      const { roles: userRoles } = jwt.verify(token, secret);
      let isCorrectRole = false;
      userRoles.forEach((role) => {
        if (roles.includes(role)) {
          isCorrectRole = true;
        }
      });
      if (!isCorrectRole) {
        return res.status(403).json({ message: 'User is not authorized' });
      }

      next();
    } catch (e) {
      {
    }
  }
};

```

					ІАЛЦ.467800.007 Д4	Арк.
						14
Зм.	Арк.	№ докум.	Підпис	Дата		

```

app.post('/projects/:projectId/taskCreate', async (req, res) => {
  try {
    const { projectId } = req.params;
    const { title, description, assignedTo, dueDate } = req.body;

    const project = await Project.findById(projectId);

    if (!project) {
      return res.status(404).json({ message: 'Project not found' });
    }

    const task = new Task({
      title,
      description,
      assignedTo,
      dueDate,
      project: projectId,
    });

    const savedTask = await task.save();

    project.tasks.push(savedTask._id);
    await project.save();

    return res.status(201).json({ message: 'Task created successfully', task: savedTask });
  } catch (error) {
    return res.status(400).json({ message: 'Error creating task', error: error.message });
  }
});

```

```

app.post('/projects/:projectId/taskUpdate/:taskId', async (req, res) => {
  try {
    const { projectId, taskId } = req.params;
    const { title, description, assignedTo, dueDate, status } = req.body;

    const project = await Project.findById(projectId);

    if (!project) {
      return res.status(404).json({ message: 'Project not found' });
    }

    const task = await Task.findById(taskId);

    if (!task) {
      return res.status(404).json({ message: 'Task not found' });
    }

    task.title = title;

```

					ІАЛЦ.467800.007 Д4	Арк.
						15
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        task.description = description;
        task.assignedTo = assignedTo;
        task.dueDate = dueDate;
        task.status = status;

        const updatedTask = await task.save();

        return res.json({ message: 'Task updated successfully', task: updatedTask });
    } catch (error) {
        return res.status(400).json({ message: 'Error updating task', error: error.message
    });
    }
});

app.get('/projects/:projectId/task/:taskId', async (req, res) => {
    try {
        const { projectId, taskId } = req.params;

        const project = await Project.findById(projectId);

        if (!project) {
            return res.status(404).json({ message: 'Project not found' });
        }

        const task = await Task.findById(taskId);

        if (!task) {
            return res.status(404).json({ message: 'Task not found' });
        }

        return res.json({ task });
    } catch (error) {
        return res.status(400).json({ message: 'Error fetching task', error: error.message });
    }
});

app.put('projects/:projectId/update', async (req, res) => {
    try {
        const { projectId } = req.params;

        const project = await Project.findById(projectId);

        if (!project) {
            return res.status(404).json({ message: 'Project not found' });
        }

        project.title = req.body.title;
        project.description = req.body.description;
        project.startDate = req.body.startDate;
    app.post('/create', async (req, res) => {
        try {

```

					ІАЛЦ.467800.007 Д4	Арк.
						16
Зм.	Арк.	№ докум.	Підпис	Дата		

```

const { title, description, startDate, endDate, createdBy } = req.body;

const project = new Project({
  title,
  description,
  startDate,
  endDate,
  createdBy,
});

const savedProject = await project.save();
res.status(200).json({ message: 'Project created successfully', project: savedProject });
} catch (error) {
  res.status(400).json({ message: 'Error creating project', error: error.message });
}
});

const express = require('express');
const app = express();
const mongoose = require('mongoose');
const Project = require('./models/Project');

const Comment = require('../models/comment');

app.post('/projects/:projectId/task/:taskId/createComment', async (req, res) => {
  const projectId = req.params.projectId;
  const taskId = req.params.taskId;
  const { author, text } = req.body;

  if (!author || !text) {
    return res.status(400).json({ message: 'Author and text are required' });
  }

  try {
    const comment = new Comment({
      projectId,
      taskId,
      author,
      text,
      timestamp: Date.now()
    });

    await comment.save();

    res.json({ message: 'Comment created successfully', comment });
  } catch (error) {
    res.status(500).json({ message: 'Failed to create comment', error });
  }
});

const User = require('../models/user');
const Role = require('../models/role');
const bcryptjs = require('bcryptjs');
const jwt = require('jsonwebtoken');

```

					ІАЛЦ.467800.007 Д4	Арк.
						17
Зм.	Арк.	№ докум.	Підпис	Дата		

```

const { validationResult } = require('express-validator');
const { secret } = require('../config');

const generateToken = (id, roles) => {
  const payload = {
    id,
    roles,
  };
  return jwt.sign(payload, secret, { expiresIn: '24h' });
};

class AuthorizationController {
  async login(req, res) {
    try {
      const { email, password } = req.body;
      const user = await User.findOne({ email });
      if (!user) {
        return res.status(400).json({ message: 'User was not found' });
      }
      const isValidPassword = bcryptjs.compareSync(password, user.password);
      if (!isValidPassword) {
        return res.status(400).json({ message: 'Invalid credentials' });
      }
      const token = generateToken(user._id, user.roles);
      return res.json({ token });
    } catch (e) {
      res.status(400).json({ message: 'Error during login' });
    }
  }

  async registration(req, res) {
    try {
      const errors = validationResult(req);
      if (!errors.isEmpty()) {
        return res.status(400).json({ message: 'Error during registration' });
      }
      const { email, name, surname, password, role } = req.body;
      const candidate = await User.findOne({ email });
      if (candidate) {
        return res.status(400).json({ message: 'Email is already registered' });
      }
      const hashedPassword = bcryptjs.hashSync(password, 7);
      const userRole = await Role.findOne({ role });
      const user = new User({ email, name, surname, password: hashedPassword, roles:
[userRole.role] });
      await user.save();
      return res.json({ message: 'User has been created' });
    } catch (e) {
      res.status(400).json({ message: 'Error during registration', error: e });
    }
  }
}

```

					ІАЛЦ.467800.007 Д4	Арк.
						18
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    }
  {
    "name": "client",
    "version": "0.1.0",
    "private": true,
    "dependencies": {
      "@emotion/react": "^11.11.0",
      "@emotion/styled": "^11.11.0",
      "@mui/material": "^5.13.0",
      "@testing-library/jest-dom": "^5.16.5",
      "@testing-library/react": "^13.4.0",
      "@testing-library/user-event": "^13.5.0",
      "formik": "^2.2.9",
      "react": "^18.2.0",
      "react-dom": "^18.2.0",
      "react-router-dom": "^6.11.1",
      "react-scripts": "5.0.1",
      "web-vitals": "^2.1.4"
    },
    "scripts": {
      "start": "react-scripts start",
      "build": "react-scripts build",
      "test": "react-scripts test",
      "eject": "react-scripts eject"
    },
    "eslintConfig": {
      "extends": [
        "react-app",
        "react-app/jest"
      ]
    },
    "browserslist": {
      "production": [
        ">0.2%",
        "not dead",
        "not op_mini all"
      ],
      "development": [
        "last 1 chrome version",
        "last 1 firefox version",
        "last 1 safari version"
      ]
    }
  }
}

const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
    });
  }
};

```

					ІАЛЦ.467800.007 Д4	Арк.
						19
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        getLCP(onPerfEntry);
        getTTFB(onPerfEntry);
    });
}
};

export default reportWebVitals;

import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();

body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen', 'Ubuntu',
  'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
  sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New', monospace;
}

* {
  box-sizing: border-box;
}

import { render, screen } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});

```

					ІАЛЦ.467800.007 Д4	Арк.
						20
Зм.	Арк.	№ докум.	Підпис	Дата		

```

});

import Authorization from './routes/authorization';

function App() {
  return <Authorization />;
}

export default App;

.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

import LoginForm from '../login/login-form/login-form';

function Authorization() {
  return (
    <div className="App">
      <LoginForm />
    </div>
  );
}

```

					ІАЛЦ.467800.007 Д4	Арк.
						21
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        </div>
    );
}

export default Authorization;

import React from 'react';
import { Switch, Route, Redirect } from 'react-router-dom';

export const useRoutes = (isAuthenticated) => {
    if (isAuthenticated) {
        return (
            <Switch>
                <Route path="/home" exact>
                    <>main page</>
                </Route>
                <Redirect to="/home" />
            </Switch>
        );
    }

    return (
        <Switch>
            <Route path="/" exact>
                <>authorization</>
            </Route>
            <Redirect to="/" />
        </Switch>
    );
};

import React from 'react';
import { makeStyles } from '@mui/styles';
import { Button, TextField, Grid, Paper } from '@mui/material';

const useStyles = makeStyles((theme) => ({
    root: {
        flexGrow: 1,
        padding: theme.spacing(3),
    },
    paper: {
        padding: theme.spacing(2),
    },
    textField: {

marginBottom: theme.spacing(2),
    },
    button: {
        marginRight: theme.spacing(2),
    },

```

					ІАЛЦ.467800.007 Д4	Арк.
						22
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    }));

function ProjectForm() {
  const classes = useStyles();

  const handleFormSubmit = (event) => {
    event.preventDefault();
    // Handle form submission logic here
  };

  return (
    <div className={classes.root}>
      <Paper className={classes.paper}>
        <form onSubmit={handleFormSubmit}>
          <Grid container spacing={2}>
            <Grid item xs={12}>
              <TextField className={classes.textField} label="Project Name" variant="outlined"
fullWidth />
            </Grid>
            <Grid item xs={12}>
              <TextField className={classes.textField} label="Description" variant="outlined"
fullWidth multiline rows={4} />
            </Grid>
            <Grid item xs={12} sm={6}>
              <TextField className={classes.textField} label="Start Date" variant="outlined"
fullWidth type="date" />
            </Grid>
            <Grid item xs={12} sm={6}>
              <TextField className={classes.textField} label="End Date" variant="outlined"
fullWidth type="date" />
            </Grid>
            <Grid item xs={12}>
              <Button className={classes.button} variant="contained" color="primary"
type="submit">
                Create Project
              </Button>
              <Button variant="outlined" color="primary">
                Cancel
              </Button>
            </Grid>
          </Grid>
        </form>
      </Paper>
    </div>

    export default ProjectForm;

```

					ІАЛЦ.467800.007 Д4	Арк.
						23
Зм.	Арк.	№ докум.	Підпис	Дата		

```

import './login-form.css';
import login from './icons/login.png';
import register from './icons/register.png';
import { useState } from 'react';
import { Formik, Field, Form } from 'formik';

import { TextField } from '@mui/material';
import Button from '@mui/material/Button';

function LoginForm() {
  const [authorizationType, setAuthorizationType] = useState();

  const authorizationTypeHandler = (type) => {
    setAuthorizationType(type);
  };

  if (authorizationType === 'login') {
    return (
      <div class="loginWrapper">
        <div class="header">Sprintman</div>
      <div class="login-form">
        <form autoComplete="off" onSubmit={() => {}}>
          <h2 style={{ textAlign: 'center' }}>Login </h2>
          <TextField
            label="Email"
            onChange={() => {}}
            required
            variant="outlined"
            color="secondary"
            type="email"
            sx={{ mb: 3 }}
            fullWidth
            value={' '}
            error={() => {}}
          />
          <TextField
            label="Password"
            onChange={() => {}}
            required
            variant="outlined"
            color="secondary"
            type="password"
            value={' '}
            error={() => {}}
            fullWidth
            sx={{ mb: 3 }}
          />
          <Button variant="outlined" color="secondary" type="submit">
            Login

```

					ІАЛЦ.467800.007 Д4	Арк.
						24
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        </Button>

        </form>
    </div>

</div>
);
}

if (authorizationType === 'register') {
    return (
        <div class="loginWrapper">
            <div class="header">Sprintman</div>
            <div class="login-block-wrapper">
                <Formik
                    initialValues={{ name: '', email: '' }}
                    onSubmit={async (values) => {
                        await new Promise((resolve) => setTimeout(resolve, 500));
                        alert(JSON.stringify(values, null, 2));
                    }}
                >
                    <Form>
                        <Field name="name" type="text" />
                        <Field name="email" type="email" />
                        <button type="submit">Submit</button>
                    </Form>
                </Formik>
            </div>
        </div>
    );
}

return (
    <div class="loginWrapper">
        <div class="header">Sprintman</div>

        <div class="login-block-wrapper">
            <div class="choice-block" onClick={() => authorizationTypeHandler('login')}>
                <img src={login} className="choice-logo" alt="login" />
                <div class="choice-text">sign up</div>
            </div>
            <div class="choice-block" onClick={() => authorizationTypeHandler('register')}>
                <img src={register} className="choice-logo" alt="login" />
                <div class="choice-text">create account</div>
            </div>
        </div>
    </div>
);
}

```

					ІАЛЦ.467800.007 Д4	Арк.
						25
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    </div>
  );
}

export default LoginForm;

.header {
  font-weight: 700;
  font-family: 'Roboto', sans-serif;
  text-transform: uppercase;
  font-size: 104px;
  width: 100%;
  text-align: center;
  user-select: none;
  letter-spacing: 10px;
  padding-top: 100px;
}

.loginWrapper {
  background-color: #c5c6c7;
  height: 100vh;
}

.login-block-wrapper {
  display: flex;
  width: 70%;
  justify-content: space-around;
  margin: 0 auto;
  margin-top: 100px;
}

.login-form {
  display: flex;
  flex-direction: column;
  width: 30%;
  justify-content: space-around;
  margin: 0 auto;
  margin-top: 100px;
}

.choice-block {
  box-shadow: rgba(17, 12, 46, 0.15) 0px 48px 100px 0px;
  border-radius: 5px;
  user-select: none;
  height: 300px;
  width: 300px;
  padding: 30px;
  transition: all 1s ease-out;
}

.choice-block:hover {

```

					ІАЛЦ.467800.007 Д4	Арк.
						26
Зм.	Арк.	№ докум.	Підпис	Дата		

```

height: 330px;
width: 330px;
cursor: pointer;
}

.choice-logo {
width: 70%;
}

.choice-text {
font-family: 'Roboto', sans-serif;
text-transform: uppercase;
font-weight: 700;
color: #656566;
font-size: 24px;
margin-top: 30px;
}

import { useState, useCallback } from 'react';

export const useHttp = () => {
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  const request = useCallback(async (url, method = 'GET', body = null, headers = {}) => {
    setLoading(true);
    try {
      if (body) {
        body = JSON.stringify(body);
        headers['Content-Type'] = 'application/json';
      }
      const response = await fetch(url, { method, body, headers });
      const data = await response.json();

      if (!response.ok) {
        throw new Error(data.message || 'Something is wrong');
      }

      setLoading(false);

      return data;
    } catch (e) {
      setLoading(false);
      setError(e.message);
      throw e;
    }
  }, []);

```

					ІАЛЦ.467800.007 Д4	Арк.
						27
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        const clearError = useCallback(() => setError(null), []);

        return { loading, request, error, clearError };
    };

import { useState, useCallback, useEffect } from 'react';

const storageName = 'userData';

export const useAuth = () => {
    const [token, setToken] = useState(null);
    const [ready, setReady] = useState(false);
    const [userId, setUserId] = useState(null);
    const [email, setEmail] = useState(null);

    const login = useCallback((jwtToken, id, email) => {
        setToken(jwtToken);
        setUserId(id);
        setEmail(email);

        localStorage.setItem(
            storageName,
            JSON.stringify({
                userId: id,
                token: jwtToken,
                email,
            })
        );
    }, []);

    const logout = useCallback(() => {
        setToken(null);
        setUserId(null);
        setEmail(null);

        localStorage.removeItem(storageName);
    }, []);

    useEffect(() => {
        const data = JSON.parse(localStorage.getItem(storageName));

        if (data && data.token) {
            login(data.token, data.userId, data.currentEmail);
        }
        setReady(true);
    }, [login]);
    return { login, logout, token, userId, ready, email };

```

					ІАЛЦ.467800.007 Д4	Арк.
						28
Зм.	Арк.	№ докум.	Підпис	Дата		

```

import {
  createContext } from
'react';

function cap() {}

export const AuthContext = createContext({
import {useState, useCallback} from 'react'

export const useHttp = () => {
  const [loading, setLoading] = useState(false)
  const [error, setError] = useState(null)

  const request = useCallback(async (url, method = 'GET', body = null, headers = {}) => {
    setLoading(true)
    try {
      if (body) {
        body = JSON.stringify(body)
        headers['Content-Type'] = 'application/json'
      }

      const response = await fetch(url, {method, body, headers})
      const data = await response.json()

      if (!response.ok) {
        throw new Error(data.message || 'Something is wrong')
      }

      setLoading(false)

      return data
    } catch (e) {
      setLoading(false)
      setError(e.message)
      throw e
    }
  }, [])

  const clearError = useCallback(() => setError(null), [])

  return { loading, request, error, clearError }
}

import './App.css'
import {AuthContext} from "../context/AuthContext"
import {useAuth} from "../hooks/auth.hook"
import {BrowserRouter as Router} from "react-router-dom"

import {useRoutes} from "../routes"

function App() {
  const {token, login, email, logout, userId} = useAuth()
  const isAuthenticated = !!token

  const routes = useRoutes(isAuthenticated)

  return (
    <div className="App">
      <AuthContext.Provider value={{
        token, login, logout, userId, isAuthenticated, email
      }}>
        <Router>
          {routes}
        </Router>
      </AuthContext.Provider>
    </div>
  )
}

```

					ІАЛЦ.467800.007 Д4	Арк.
						29
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    </div>
  );
}

import './Loader.css'
import {LoadingOutlined} from '@ant-design/icons'

const Loader = ()=>{
  return(
    <div className="loaderBlock">

<LoadingOutlined style={{ fontSize: '156px', color: 'gray' }}/>
    </div>
  )
}

export default Loader

.loaderBlock{
  width: 100%;
  height: 100%;
  display: flex;
  justify-content: space-around;
  align-items: center;
}

import './HeaderBlock.css'
import { Avatar } from 'antd';
import {useState, useEffect, useCallback} from "react"
import {useHttp} from "../../hooks/http.hook"
import { UserOutlined } from '@ant-design/icons';

const HeaderBlock = (props)=>{
  const {request} = useHttp()
  const [email, setEmail] = useState()
  console.log(email);

  const getInfo = useCallback(async () => {
    try {
      const data = await request('/api/info/getInfo', 'POST', {id:
localStorage.getItem('userData')})
      setEmail(data.email)
    } catch (e) {}
  }, [request])
  useEffect(()=>{
    getInfo()
  }, [getInfo])
  if(!email){
    return(
      <div>qq</div>
    )
  }
  return(
    <div className="headerBlock">

```

					ІАЛЦ.467800.007 Д4	Арк.
						30
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        <div className="upperPartHeader" />
        <div className="bottomPartHeader">
            <div className="avatarBox">
                <Avatar shape="square" size={184} icon={<UserOutlined />} />
            </div>
            <div className="headerEmail">
                {email}
            </div>

        </div>
    </div>
)
}

```

```
export default HeaderBlock
```

```

.headerBlock {
  width: 100%;
  height: 200px;
  margin-top: 30px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.5);
  position: relative;
}

```

```

.upperPartHeader {
  width: 100%;
  height: 50%;
  background-color: #ababab;
}

```

```

.avatarBox {
  margin-left: 10px;
  margin-top: -90px;
}

```

```

.bottomPartHeader {
  display: flex;
  font-size: 34px;
}

```

```

.headerEmail {
  margin-top: 20px;
  margin-left: 20px;
}

```

```
import { useState, useEffect, useContext, useCallback } from 'react'
```

					ІАЛЦ.467800.007 Д4	Арк.
						31
Зм.	Арк.	№ докум.	Підпис	Дата		

```

import './MainPage.css'
import { AuthContext } from '../../context/AuthContext.js'
import InfoBlock from '../InfoBlock/InfoBlock'
import TableBlock from '../TableBlock/TableBlock'
import FieldBlock from '../FieldBlock/FieldBlock'
import TeamWeek from '../TeamWeek/TeamWeek'

import { useHttp } from '../../hooks/http.hook'

const MainPage = (props) => {
  const auth = useContext(AuthContext)
  const [tableInfo, setTableInfo] = useState(null)
  const { request } = useHttp()

  const getTable = useCallback(async () => {
    try {
      const data = await request('/api/table')
      setTableInfo(data)
    } catch (e) {}
  }, [request])

  useEffect(() => {
    getTable()
  }, [getTable])

  const logoutHandler = (event) => {
    event.preventDefault()
    auth.logout()
  }

  return (
    <div className="mainBlock">
      <div className="mainWrapper">
        <InfoBlock />
        <TableBlock tableInfo={tableInfo} />
      </div>
      <div className="mainWrapper">
        <FieldBlock />
        <TeamWeek />
      </div>
      <li style={{marginTop: "15px"}}>
        <a href="/" onClick={logoutHandler}>
          Вийти
        </a>
      </li>
    </div>
  )
}

```

					ІАЛЦ.467800.007 Д4	Арк.
						32
Зм.	Арк.	№ докум.	Підпис	Дата		

```
    )
}

export default MainPage
  .mainBlock {
    font-family: "Roboto";
    width: 90%;
    margin: 0 auto;
  }

.mainWrapper {
  width: 100%;
  margin-top: 15px;
  display: flex;
  justify-content: space-between;
}

@media screen and (max-width: 1570px) {
  .mainWrapper{
    display: inline-block;
  }
}
```

					ІАЛЦ.467800.007 Д4	Арк.
						33
Зм.	Арк.	№ докум.	Підпис	Дата		