МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____Аліна САВЧЕНКО
«___»_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ **“БАКАЛАВР”**

Тема: «Вебзастосунок для системи управління проектами з використанням фреймворку Angular»

Виконавець: Єлизавета ВАСИЛЕНКО

Керівник: к.т.н., доцент кафедри КІТ Олена ТОЛСТІКОВА

Нормоконтролер: к.т.н., доцент Вікторія СИДОРЕНКО

КИЇВ 2024

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних наук та технологій*

Кафедра *комп'ютерних інформаційних технологій*

Спеціальність *122 «Комп'ютерні науки»*

Освітньо-професійна програма *«Інформаційні технології проектування»*

ЗАТВЕРДЖУЮ:
завідувач кафедри КІТ
_____Аліна САВЧЕНКО
(підпис)
«____»_____2024 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Василенко Єлизавети Сергіївни

(прізвище, ім'я, по батькові здобувача вищої освіти в родовому відмінку)

1. Тема кваліфікаційної роботи: «Вебзастосунок для системи управління проектами з використанням фреймворку Angular», затверджена наказом ректора від «05» квітня 2024 р. № 517/ст.

2. Термін виконання роботи: з 06 травня 2024 року по 16 червня 2024 року.

3. Вихідні дані до роботи: середовище розробки Visual Studio Code, платформа Docker, мова програмування TypeScript, фреймворки Angular та Express.js, середовище Node.js та база даних MongoDB.

4. Зміст пояснювальної записки: 1) Огляд та аналіз систем управління проектами. 2) Технології та інструменти розробки. 3) Вебзастосунок для системи управління проектами.

5.Перелік обов'язкового ілюстративного матеріалу: слайди презентації MS PowerPoint: 1. Актуальність та мета 2. Об'єкт, предмет дослідження та наукова новизна, 3. Задачі кваліфікаційної роботи, 4. Технології та інструменти розробки, 5. Демонстарція роботи вебзастосунку.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1.	Аналіз предметної області.	06.05.2024–08.05.2024	
2	Збір та аналіз інформації за темою кваліфікаційної роботи.	09.05.2024–11.05.2024	
3	Написання 1 розділу, представлення керівнику.	12.05.2024–15.05.2024	
4	Визначення основних вимог роботи та використаних технологій. Написання 2 розділу, представлення керівнику.	16.05.2024–19.05.2024	
5	Розробка клієнтської та серверної частин застосунку. Написання 3 розділу, представлення керівнику.	20.05.2024–25.05.2024	
6	Оформлення та друк пояснювальної записки.	26.05.2024–30.05.2024	
7	Підготовка демонстраційного матеріалу та доповіді.	31.05.2024–07.06.2024	

7. Дата видачі завдання «06» травня 2024 р.

Керівник кваліфікаційної роботи _____ Олена ТОЛСТІКОВА
(підпис керівника)

Завдання прийняв до виконання _____ Єлизавета ВАСИЛЕНКО
(підпис здобувача вищої освіти)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему: «Вебзастосунок для системи управління проектами з використанням фреймворку Angular» містить: 55 сторінок, 40 рисунків та 17 інформаційних джерел.

Об’єкт дослідження – розробка вебзастосунку системи управління проектами.

Предмет дослідження – вебзастосунок на базі фреймворку Angular для оптимізації процесів управління проектами.

Мета роботи – підвищити ефективність управління проектами шляхом впровадження нового вебзастосунку, з використанням фреймворку Angular, що спростить процеси планування, виконання та контролю проектів.

Методи дослідження: аналіз літературних джерел та предметної області, дослідження ринку, проектування та розробка застосунку.

Наукова новизна полягає у впровадженні нової системи управління проектами на основі фреймворку Angular. Що включає реалізацію взаємодії в реальному часі через WebSocket, статичну типізацію за допомогою TypeScript та зручний користувацький інтерфейс.

Ключові слова: ВЕБЗАСТОСУНОК, СИТЕМА УПРАВЛІННЯ ПРОЕКТАМИ, ANGULAR, КЛІЄНТ, СЕРВЕР, HTTP, WEBSOCKET, TYPESCRIPT, MONGODB, ФРЕЙМВОРК.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

API	–	Application Programming Interface
SPA	–	Single-page application
HTTP	–	HyperText Transfer Protocol
RxJS	–	Reactive Extensions for JavaScript
CLI	–	Command Line Interface
СКБД	–	Система Керування Базами Даних
CRUD	–	Create, Read, Update, Delete
MVC	–	Model-View-Controller
VS Code	–	Visual Studio Code
HTML	–	HyperText Markup Language
DOM	–	Document Object Model

ВСТУП

Зважаючи на нинішню цифрову реальність, де щодня з’являються нові ідеї та проекти, важливим фактором успіху стає ефективне керування цими ініціативами. Незалежно від розміру чи складності, кожен проект потребує чіткого планування та ефективного управління, щоб досягти поставлених цілей.

Актуальність теми полягає у потребі в спеціалізованих інструментах, які сприятимуть оптимізації та спрощенню робочих процесів. Використання застосунків для управління проектами покращує процеси планування, виконання та моніторингу завдань, дозволяючи зберігати всю потрібну інформацію в одному місці, доступному з будь-якого пристрою. Такі інструменти повинні відповідати потребам користувачів у функціональності, мати інтуїтивно зрозумілий інтерфейс та забезпечувати швидкий доступ до необхідної інформації.

Вивчення та аналіз існуючих додатків з управління проектами дозволить виявити їхні переваги та недоліки. Використання сучасних технологій, зокрема фреймворку Angular, може значно полегшити розробку та впровадження вебзастосунку для управління проектами. Angular забезпечує стабільність та швидкість роботи вебдодатків, а також дозволяє побудувати динамічні та інтерактивні інтерфейси, що сприяє покращенню користувацького досвіду.

Метою кваліфікаційної роботи є розробка вебзастосунку на основі фреймворку Angular для системи управління проектами, спрямованого на полегшення та оптимізацію процесів планування, виконання та моніторингу завдань. Для досягнення поставленої мети необхідно вирішити наступні **завдання дослідження**:

- аналіз існуючих вебзастосунків та програм для управління проектами з метою визначення їхніх переваг, недоліків та особливостей;
- вибір оптимальних інструментів та технологій розробки;

- дослідження особливостей фреймворку Angular;
- створення серверної логіки, бази даних та API;
- розробка клієнтської частини вебзастосунку на базі фреймворку Angular;
- оцінка функціонування розробленого вебзастосунку.

Об’єктом дослідження кваліфікаційної роботи є розробка вебзастосунку системи управління проектами.

Предметом дослідження кваліфікаційної роботи є вебзастосунок на базі фреймворку Angular для оптимізації процесів управління проектами**.**

Методи дослідження, використані для досягнення поставленої мети та виконання завдань, включають:

− аналіз літературних джерел та предметної області: дослідження джерел, що стосуються управління проектами, веброзробки та технологій у цій галузі, а також фреймворку Angular;

− дослідження ринку: аналіз існуючих вебзастосунків та програм для управління проектами з метою отримання інформації про їх функціональні можливості, потреби користувачів, недоліки та переваги;

− проектування та розробка застосунку: розробка вебзастосунку на основі отриманих знань, що включає в себе, створення архітектури, інтерфейсу користувача, розробку клієнтської та серверної логіки.

Наукова новизна полягає у впровадженні нової системи управління проектами на основі фреймворку Angular. Що включає реалізацію взаємодії в реальному часі через WebSocket, статичну типізацію за допомогою TypeScript та зручний користувацький інтерфейс.

Практичне значення отриманих результатів: результати кваліфікаційної роботи можуть бути використані для вдосконалення процесів управління проектами та завданнями в різних сферах діяльності, таких як інформаційні технології, будівництво, маркетинг тощо.

РОЗДІЛ 1
ОГЛЯД ТА АНАЛІЗ СИСТЕМ УПРАВЛІННЯ ПРОЕКТАМИ

1.1. Основні поняття веброзробки

Розробка будь-якого вебзастосунку – це комплексний процес створення програмного забезпечення, яке буде доступне для використання через веббраузер. Найчастіше такі застосунки впроваджуються для виконання певних завдань або надання послуг користувачам. Оскільки вебзастосунки мають на меті взаємодію з користувачами, це означає, що вони повинні бути зручними у використанні, надійними і ефективними.

Вебсторінка – це документ, доступний для перегляду за допомогою веббраузера. Такий документ має бути написаним за допомогою HTML та мати відповідний до мови розмітки гіпертексту синтаксис. Кожна вебсторінка має певну адресу, за допомогою якої можна отримати до неї доступ, а також містить у собі різноманітні елементи. Такими елементами можуть бути блоки тексту, картинки, відео, посилання на інші сторінки тощо.

Вебсайт або просто сайт – це сукупність пов’язаних між собою вебсторінок. Кожен сайт має своє доменне ім’я, за яким його можна знайти і отримати до нього доступ і, як правило, належить одному власнику або організації. Зазвичай сайти поділяються на дві великі категорії:

- Статичні: складаються з вебсторінок, вміст яких залишається постійним, а зміна у відображенні контенту відбувається лише за прямого втручання компетентної особи. Найпопулярнішими варіантами їх використання є блоги та сайти-портфоліо.

– Динамічні: автоматично оновлюють дані та генерують вебсторінки, реагуючи на дії користувача або відповіді від сервера. Це забезпечує інтерактивність та адаптивність контенту до користувацьких запитів і змін. Прикладами таких сайтів можуть бути соціальні мережі, інтернет-магазини або новинні портали.

Наприклад якщо взяти будь-який вебсайт з продажу товарів, то вебсторінками можуть бути домашня сторінка, сторінка товару, сторінка для оформлення замовлення тощо. Набір таких вебсторінок, об’єднаних особливими інтерактивними елементами (зазвичай клікабельні блоки тексту), і являє собою вебсайт. Інтерактивні елементи забезпечують навігацію між сторінками та взаємодію користувача з контентом, що значно підвищує зручність використання сайтом.

Щодо вебзастосунку, то він включає не тільки вебсторінки та інтерактивні елементи, але й більш складну логіку та функціональність. Вебзастосунок або вебдодаток являє собою програмне забезпечення доступ до якого здійснюється через браузер. Фактично це більш інтерактивна і складна форма вебсайту. У вебзастосунках активно використовується комбінація сценаріїв на стороні сервера та сценаріїв на стороні клієнта для представлення інформації. Вони також майже завжди включають бази даних задля збереження інформації і мають змогу взаємодіяти з іншими сервісами через API. [1, 2]

Однією з основних форм представлення вебзастосунків є SPA, односторінковий вебдодаток. Суть SPA полягає в тому, що весь необхідний контент завантажується одночасно на початку сесії, а подальші взаємодії і зміни здійснюються без перезавантаження сторінки. Це значно підвищує швидкість і зручність використання додатка, оскільки користувачеві не потрібно чекати завантаження нових сторінок сервером. Тим не менш, такі додатки мають і свої недоліки, зокрема початковий час завантаження і складнощі з SEO-оптимізацією

(Search Engine Optimization). Проте, нині існує багато способів мінімізувати ці недоліки, наприклад, через оптимізацію ресурсів. [3]

Такі вебзастосунки створюються за допомогою JavaScript фреймворків, найпопулярнішими серед яких є Angular, React та Vue.js. Використання фреймворків значно пришвидшує і спрощує процес розробки, впровадження і підтримки вебдодатків. Розробникам надаються готові рішення для багатьох типових задач, що дозволяє зосередитись на логіці додатка та покращенні користувацького досвіду.

1.2. Клієнт-серверна архітектура в межах вебзастосунку

1.2.1. Основи клієнт-серверної архітектури

“Клієнт-сервер” – це архітектурна модель, у якій один або кілька клієнтських пристроїв запитують дані від сервера. Завдяки такому підходу досягається розділення обов’язків та операцій, а також можливість одразу декільком користувачам отримати доступ і використовувати один і той самий ресурс.

Клієнтом є додаток або пристрій, який здійснює запити до сервера. Це може бути браузер або мобільний застосунок із зручним інтерфейсом, який відправляє запити на сервер для здійснення певних операцій, отримання відповіді та її подальшої обробки. Сервер, відповідає за обробку цих запитів, виконує необхідні операції та повертає відповідь клієнту (рис 1.1).

Рис. 1.1. Клієнт-серверна архітектура

Часто така архітектура включає в себе базу даних, яка зберігає та обробляє дані, необхідні для роботи вебзастосунку. Сервер взаємодіє з базою, виконуючи операції запису, оновлення, видалення та отримання інформації, а потім передає ці дані клієнту.

Клієнт-серверне спілкування здійснюється за допомогою мережевих протоколів передачі даних. Найпоширенішим з них є HTTP, який визначає правила обміну інформацією. Протокол HTTPS (HyperText Transfer Protocol Secure) додає захищене з'єднання, забезпечуючи шифрування даних для більшої безпеки. [4, 5]

1.2.2. Переваги і недоліки клієнт-серверної архітектури

Кожна технологія, модель або ж підхід до розробки мають як сильні, так і слабкі сторони. Клієнт-серверна архітектура не є винятком.

Серед переваг даної моделі слід зазначити:

− розподілену структуру: Чіткий поділ на клієнтську та серверну частини, в значній мірі зменшує навантаження на кожну з сторін. Такий підхід дозволяє зосередитись на розробці та оптимізації кожної частини окремо;

− безпеку даних: За допомогою сервера є можливість керувати доступністю даних та виконувати автентифікацію користувачів, що значно покращує рівень безпеки;

− масштабованість: У разі необхідності збільшення потужності сервера або обробки більшої кількості запитів від клієнтів, сервери можна легко масштабувати, розширювати та адаптувати під нові вимоги.

Недоліками клієнт-серверної архітектури є:

− залежність від мережі: Додаток буде працювати швидко і ефективно тільки при якісному і постійному мережевому з’єднанні. Проблеми з мережею, можуть призвести до неповноцінного функціонування застосунка;

− грошові витрати: Не залежно від масштабу проекту або застосунку для впровадження та утримання серверної інфраструктури потрібні фінансові ресурси;

− залежність клієнта від серверу: Якщо сервер виходить з ладу або стає недоступним, клієнти втрачають можливість зобразити та опрацювати дані. Фактично клієнт частково або повністю втрачає працездатність, що може негативно вплинути на користувацький досвід і знизити загальну продуктивність додатка.

Незважаючи на недоліки, клієнт-серверна архітектура є однією з найпоширеніших моделей для створення надійних і масштабованих вебдодатків. Беручи до уваги як недоліки, так і переваги, можна зробити висновок, що саме така модель є влучним вибором для розробки вебзастосунку для управління проектами.

1.2.3. HTTP та WebSocket

Однією з основних задач вебзастосунків є ефективна і швидка взаємодія клієнта з сервером для обміну даними та виконання певних операцій. Для здійснення такої взаємодії існують різні протоколи, але основними є HTTP та WebSocket, кожен з яких має свої особливості.

HTTP – це односпрямований протокол для передачі гіпертексту, який дозволяє клієнтам, наприклад веббраузерам, і серверам обмінюватись інформацією. Саме він лежить в основі обміну даними в Інтернеті і підтримує такі методи як, GET, POST, PUT, DELETE та інші, які визначають тип операції, що виконується над ресурсом.

HTTP працює за шаблоном “запит-відповідь”, тобто клієнт робить запит, а сервер надсилає відповідь. Разом з відповіддю повертаються дані (тіло повідомлення) і додаткова інформація про запит, наприклад, статусний код (показує, чи успішно оброблено запит) та заголовки (містять метадані про запит або відповідь). Для кожного запиту відкривається короткочасне підключення до сервера, яке потім закривається (рис. 1.2). Варто зазначити, що HTTP є протоколом без збереження стану, тому кожен запит виконується незалежно від попередніх і не має доступу до їхньої інформації.

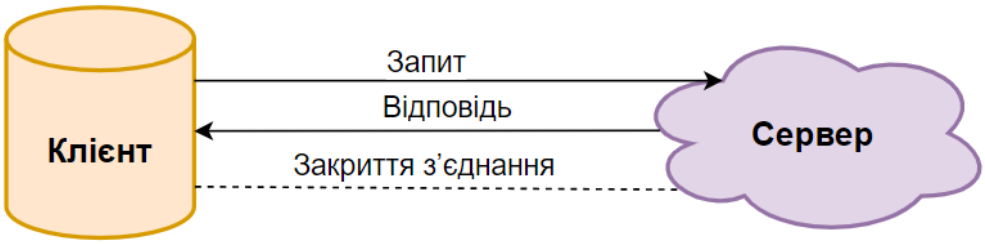

Рис. 1.2. HTTP з'єднання

Принцип роботи протоколу HTTP:

1. Клієнт ініціює запит на сервер.
2. Відкривається короткочасне з'єднання.
3. Сервер отримує та обробляє запит.
4. Сервер надсилає відповідь клієнту, разом з тілом повідомлення і додатковою інформацією.
5. З'єднання закривається.

WebSocket – подібно до HTTP є протоколом, який дозволяє обмінюватись інформацією. Його ключовою відмінністю є те, що це двосторонній протокол, який встановлює постійне з'єднання між клієнтом і сервером (рис. 1.3). За допомогою такого з'єднання є можливість передавати дані в обох напрямках в режимі реального часу. Якщо його розірвати з одного боку, та передача даних припиняється, і зв'язок між клієнтом і сервером втрачається. Це забезпечує значно швидшу взаємодію і зменшує затримки, оскільки немає потреби відкривати і закривати з'єднання для кожного запиту. [6]

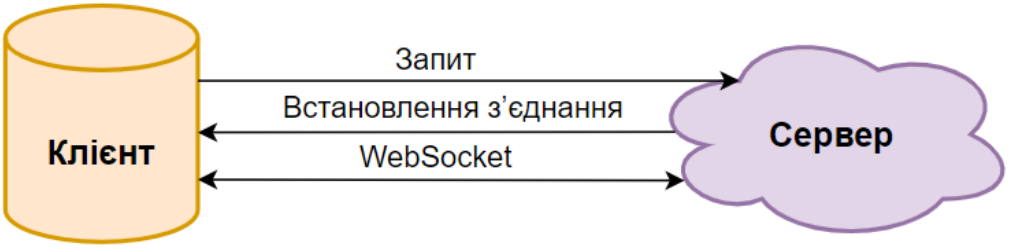

Рис. 1.3. WebSocket з'єднання

Використання WebSocket є актуальним у вебзастосунках, в яких важливий швидкий обмін даними та взаємодія в реальному часі.

Принцип роботи протоколу WebSocket:

1. Клієнт надсилає запит на сервер для встановлення WebSocket-з’єднання.
2. Встановлення з’єднання.
3. Двосторонній обмін даними.
4. Закриття з’єднання.

Підсумовуючи, можна зробити висновок, що у контексті вебзастосунку для управління проектами, вибір між HTTP та WebSocket залежить від специфічних вимог до взаємодії з користувачем. HTTP протокол підходить для більшості стандартних операцій, таких як завантаження сторінок, відправка форм, отримання та редагування даних.

WebSocket, натомість, є гарним вибором для реалізації функцій, які потребують миттєвого оновлення та безперервного обміну інформацією між клієнтом і сервером. Це, наприклад, може бути редагування задач, зміна їх розташування, оновлення статусів тощо.

Для ефективного та зручного користувацького досвіду, важливо вміти поєднувати різні підходи. Використання HTTP для базових операцій забезпечує стабільність і надійність, тоді як WebSocket додає інтерактивності та динамічності.

1.3. Огляд існуючих аналогів застосунку для управління проектами

Успішна реалізація будь-якого проекту, ідеї чи бізнесу це кропіткий і послідовний процес. Для управління таким процесом існують різноманітні програмні засоби, які відрізняються за своїми можливостями та функціональністю. Огляд та аналіз існуючих аналогів таких додатків дозволить отримати уявлення про різноманітність доступних рішень, їх недоліки та можливості для покращення.

1.3.1. JIRA

JIRA є потужною платформою для управління проектами та завданнями, розробленою компанією Atlassian (рис. 1.4). За допомогою даної платформи командам різного розміру, великим компаніям або ж окремими користувачам надається можливість керувати проектами та відстежувати процес роботи. Варто зазначити, що з моменту представлення, JIRA набула великої популярності і вже багато років займає лідуючі позиції в категорії інструментів для управління проектами. [7]

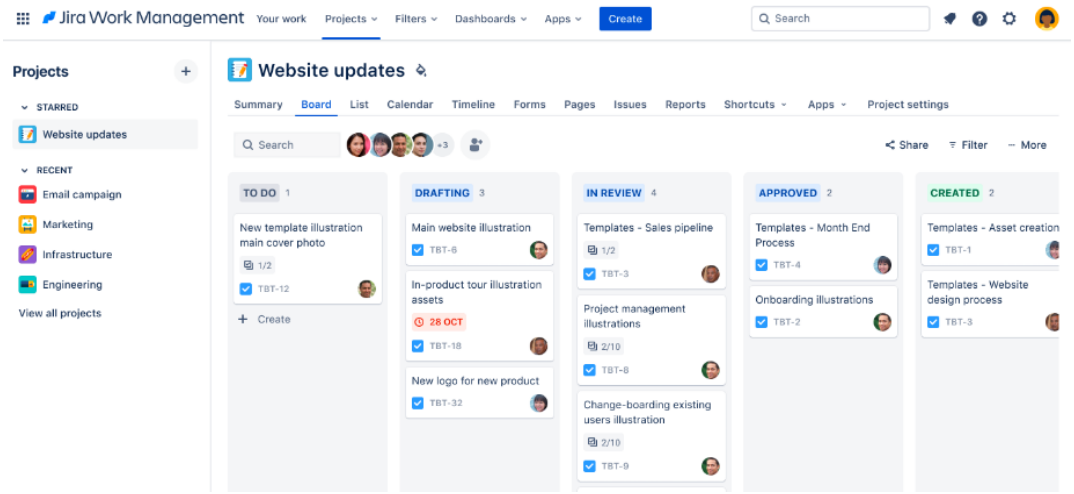

Рис. 1.4. Робоча область у JIRA

Переваги JIRA

Гнучкість та велика кількість налаштувань: Дана платформа надає можливість працювати та керувати одразу кількома проектами, що дійсно важливо для великих компаній, адже є централізоване управління всіма робочими процесами. Також дана платформа має багато опцій для персоналізації та автоматизації, та може бути налаштована під особливі потреби проекту.

Різноманітні розширення та інтеграція зі сторонніми сервісами: Для покращення і розширення функціональності застосунку, в JIRA надається підтримка великої кількості плагінів, а також можливість інтеграції з іншими

сервісами, такими як Confluence, Slack, GitHub та іншими. Це дозволяє створювати більш інтегровану та ефективну робочу середу.

Недоліки JIRA

Вартість використання: Для невеликих команд до десяти людей, у яких немає потреби в додатковій інтеграції з платними сервісами, існує задовільний безкоштовний план. Проте для більших компаній та команд, JIRA може стати дійсно дорогим інструментом.

Складність в налаштуванні та навчанні: Через широкі можливості у налаштуванні даної системи, початкове освоєння може зайняти певний час і ресурси.

1.3.2. Monday.com

Monday.com – це інструмент для управління проектами, який сприяє ефективній командній роботі, дозволяє відслідковувати виконання завдань та оптимізувати робочі процеси (рис. 1.5). Ця платформа має приємний дизайн і зручний користувацький інтерфейс, що робить її популярним вибором серед різноманітних команд та організацій. [8]

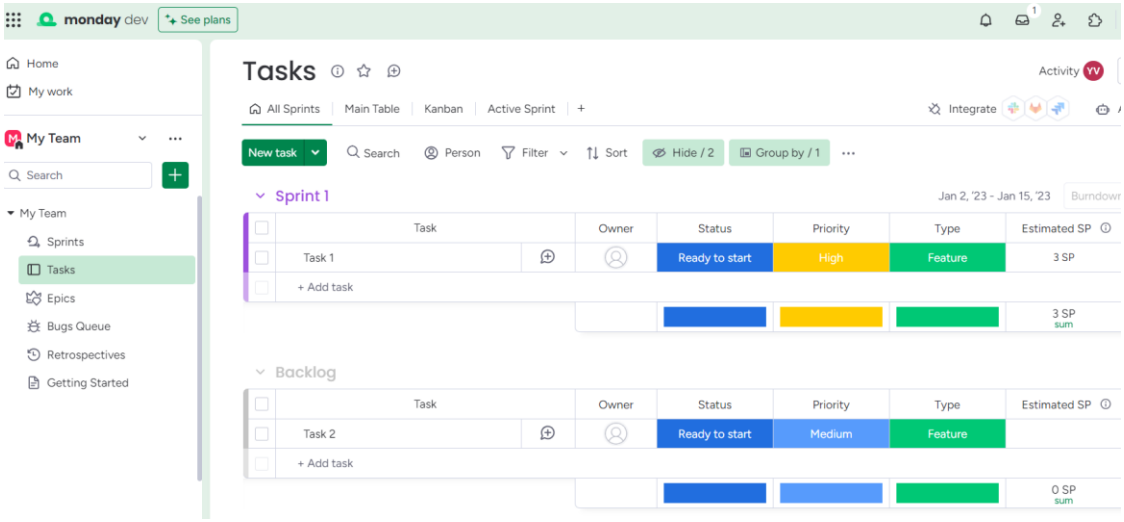

Рис. 1.5. Робоча область у Monday.com

Переваги Monday.com

Зручний користувацький інтерфейс: Завдяки приємному і зрозумілому інтерфейсу, додаток дуже легкий у освоєнні навіть для користувачів без досвіду. Яскраві та інтерактивні елементи допомагають швидко вивчити можливості даної платформи та налаштувати її під свої потреби.

Вбудовані засоби для звітності та аналітики: Використовуючи Monday.com, є можливість будувати звіти та проводити аналіз продуктивності команди. Це дозволяє відстежувати прогрес, виявляти проблеми та приймати обґрунтовані рішення для покращення роботи.

Недоліки Monday.com

Зменшення продуктивності при великих навантаженнях: Велика кількість проектів, користувачів та завдань можуть призвести до сповільнення роботи платформи, що може вплинути на загальну продуктивність команди.

Вартість: Безкоштовний план наявний, але лише для окремого користувача або ж пари користувачів. Додаткові функції, такі як відстеження часу, аналітика та розширені інтеграції, доступні лише у платній версії, що може збільшити витрати на використання платформи.

1.3.3. Miro

Miro – це онлайн-платформа, розроблена для візуальної співпраці. Вона дозволяє користувачам взаємодіяти один з одним в режимі реального часу, використовуючи інтерактивні дошки, схеми, діаграми та інші інструменти для візуалізації (рис. 1.6). Завдяки своїй гнучкості та різноманітним функціям, Miro є популярним вибором серед команд, особливо якщо їх учасники знаходяться в різних куточках світу. [9]

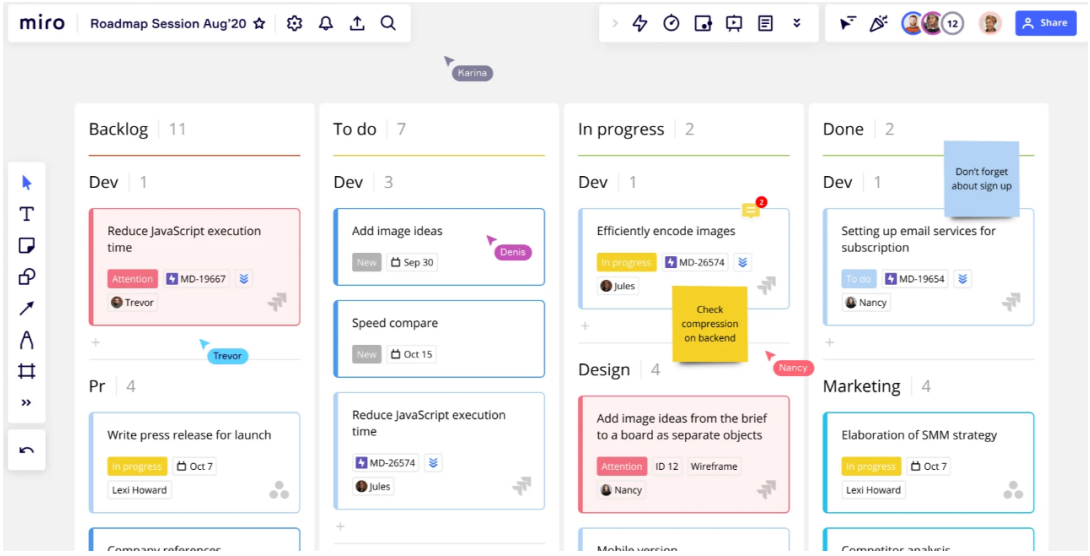

Рис. 1.6. Робоча область у Miro

Переваги Miro

Взаємодія у реальному часі та широкі можливості візуалізації: Користувачам надається великий вибір різноманітних шаблонів та інструментів, для створення діаграм, схем, дошок, нотаток тощо. А можливість одночасного редагування та коментування дозволяє командам працювати над проектами одночасно, що сприяє кращій взаємодії та швидкому прийняттю рішень.

Зручність та гнучкість: Доступ до платформи можна отримати, не лише за допомогою веббразуера, а ще й за допомогою мобільного додатку для iOS та Android, що дозволяє користувачам працювати з будь-якого місця. Різноманітні можливості для налаштування підвищують ефективність роботи і роблять користувацький досвід кращим.

Недоліки Miro

Ресурсоємність та продуктивність: Під час взаємодії з великими складними дошками або схемами Miro може вимагати значних обчислювальних ресурсів. Це може спричинити навантаження на пристрій, що може сповільнювати роботу користувачів і викликати затримки в обробці даних.

Ціна: Подібно до вже розглянутих вище платформ, Miro пропонує обмежений набір функцій у безкоштовному плані і надає можливість працювати лише з трьома дошками. Для збільшення доступних для редагування дошок та додаткових функцій, необхідно придбати платний план, який є доволі дорогим для невеликих команд чи окремих користувачів.

1.3.4. Аналіз існуючих рішень

Отже, розглянувши кілька найпопулярніших платформ для керування проектами, можна зробити висновок, що, незважаючи на їхню ефективність, кожна має свої недоліки. Всі додатки мають обмежену функціональність у безкоштовному варіанті, що, звісно, не є проблемою для великих компаній, але може стати такою для окремого користувача або невеликих команд.

Із переваг JIRA, варто зазначити гнучкість, різноманітні розширення та інтеграцію зі сторонніми сервісами. Платформа дозволяє створювати власні робочі процеси, налаштовувати поля, екрани, сповіщення та інші параметри під специфічні потреби проекту. Недоліками є вартість використання і складність в початковому налаштуванні та навчанні.

Щодо Monday.com, то серед переваг можна виділити зручний користувацький інтерфейс, гарний і сучасний дизайн та вбудовані засоби для звітності та аналітики. Завдяки цьому користувачі доволі швидко освоюють платформу. Однак вона може мати проблеми з продуктивністю при великих навантаженнях, а також вартість додаткових функцій є значною, особливо для невеликих команд.

Плюсами Miro є можливість взаємодії у реальному часі та широкий вибір інструментів візуалізації. Ця платформа дозволяє користувачам створювати інтерактивні дошки, схеми та діаграми, що робить її незамінною для команд, які працюють над складними проектами або ідеями. Проте, Miro також має свої недоліки. Платформа може бути ресурсоємною, що впливає на продуктивність при

роботі з великими і складними дошками, а також має високу вартість для повного доступу до функцій.

Виходячи з аналізу цих додатків, можна виділити кілька ключових аспектів, які слід врахувати при розробці власного застосунку для управління проектами. Він повинен мати зручний інтерфейс, привабливий дизайн, і включати набір необхідних функцій для управління проектами. Важливо, щоб застосунок був доступним для кожного користувача, незалежно від їхнього рівня технічних знань, і забезпечував стабільну роботу навіть при високому навантаженні.

1.4. Висновок до розділу 1

У першому розділі кваліфікаційної роботи було розглянуто основи клієнт-серверної архітектури та проведено аналіз систем управління проектами.

Початково було висвітлено основні поняття веброзробки, такі як визначення вебсторінки, вебсайту та вебзастосунку в цілому.

Ознайомлення з основами клієнт-серверної архітектури включало в себе пояснення її основних принципів, переваг та недоліків. Зокрема, було розглянуто розподіл обов’язків між клієнтом і сервером.

Особлива увага була приділена протоколам HTTP та WebSocket. Після розгляду обох протоколів, можна зробити висновок, що для досягнення ефективності роботи вебзастосунку важливо знати і вміти поєднувати переваги кожного з них. HTTP є ідеальним вибором для стандартних CRUD-операцій, тоді як WebSocket забезпечує миттєвий обмін даними в режимі реального часу.

В рамках першого розділу також було оглянуто існуючі аналоги застосунку для управління проектами, такі як JIRA, Monday.com та Miro. Аналіз цих додатків дозволив виявити їхні сильні та слабкі сторони і зрозуміти основні аспекти, якими має володіти власний вебзастосунок. Це дозволить розробити зручний, функціональний та доступний для користувачів застосунок, який забезпечуватиме стабільну роботу та ефективне управління проектами.

РОЗДІЛ 2
ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ РОЗРОБКИ

2.1. Фреймворк Angular

Angular, створений і підтримуваний Google, є потужним фреймворком JavaScript. Фреймворк – це набір інструментів, бібліотек та спеціальних правил, який використовується для створення застосунків. Angular, являючись фреймворком, надає широкі можливості для розробки додатків, а саме SPA та мобільних застосунків. Він написаний за допомогою TypeScript і забезпечує статичну типізацію, що підвищує надійність коду та полегшує його супровід. Компанії, такі як Google, PayPal, Upwork, IBM, Microsoft та Samsung, активно використовують даний інструмент у своїх проектах.

Фреймворк, відомий своєю чіткою структурою та впровадженням найкращих патернів та парадигм програмування і проектування. Наприклад, засновники Angular пропагують архітектурний підхід Model-View-Controller. MVC ділить вебпрограму на три окремі частини: Model (модель), View (представлення) та Controller (контролер) (рис. 2.1). Модель відповідає за управління даними програми. Це може включати логіку для отримання, зберігання та маніпулювання даними, які відображають стан програми. Представлення займається відображенням даних користувачеві. Контролер обробляє користувацькі запити, взаємодіє з моделлю та визначає, які дані повинні бути відображені у представленні. Такий підхід забезпечує структурований підхід до розробки вебзастосунків, підвищуючи ефективність роботи та надійність кінцевого продукту.

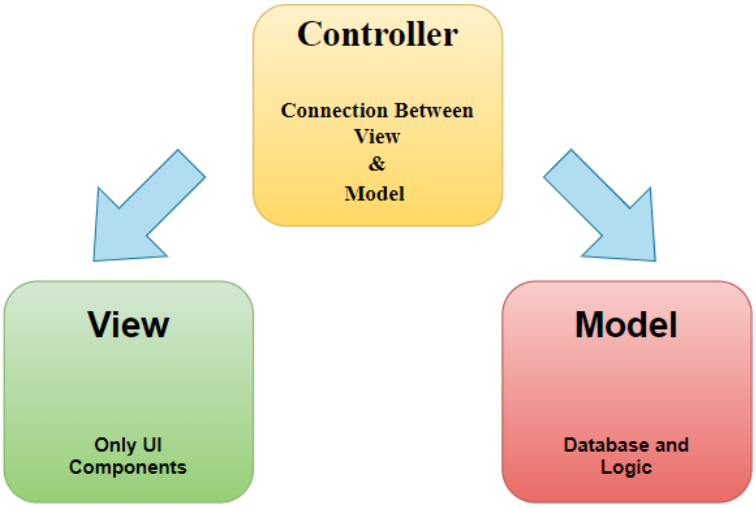

Рис. 2.1. Архітектура MVC

В Angular широко застосовується реактивне програмування – парадигма асинхронного програмування, орієнтована на обробку потоків даних та їх розповсюдження. Це дозволяє емітувати (випромінювати) потік даних або значень з одного джерела під назвою Observable, а іншим джерелам, що називаються Observer, перехоплювати цей потік даних через процес підписки та реагувати на нього.

Цей патерн Observable/Observer значно спрощує виявлення складних змін та оновлення даних. За допомогою реактивного програмування є можливість ефективно керувати асинхронними операціями та реагувати на зміни в реальному часі, що забезпечує більшу динамічність Angular-застосунків. Оскільки JavaScript не має вбудованої підтримки реактивного програмування, для його реалізації потрібна додаткова бібліотека. Однією з таких бібліотек є RxJS.

Найчастішими випадками використання RxJS є обробка HTTP-запитів, що дозволяє зручно отримувати дані з сервера та оновлювати інтерфейс відповідно до нової інформації.

Розглянемо приклад методу getBoards() (рис. 2.2), який демонструє виконання і обробку запиту для отримання списку створених користувачем проектних дошок.

Спочатку виконується GET-запит до сервера, який повертає Observable. Метод subscribe() використовується для підписки на отриманий Observable, що дозволяє обробляти потік даних (результат запиту). У разі успішного виконання запиту, метод отримує масив об’єктів boards. Якщо запит зазнає невдачі, метод виводить повідомлення про помилку в консоль.

```
getBoards(): void {
  this.http.get<BoardInterface[]>(`${environment.apiUrl}boards/`).subscribe({
    next: (boards) => {
      this.boards$.next(boards);
    },
    error: (err) => console.log(err)
  });
}
```

Рис. 2.2. Метод getBoards()

Angular є складним фреймворком з великим набором можливостей для розробки вебзастосунків. Він пропонує широкий спектр функцій, що дозволяють створювати як прості, так і надзвичайно складні додатки. Нижче наведено основні особливості Angular.

Компонентна архітектура

Компонент це – фундаментальний будівельний блок програми. Кожен компонент включає в себе шаблон, дані та поведінку представлення, що дозволяє створювати ізольовані, повторно використовувані частини інтерфейсу. Компоненти дозволяють організувати код у зрозумілий і підтримуваний спосіб, забезпечуючи чітку структуру додатка.

Двостороннє зв’язування даних

Двостороннє зв’язування даних це – механізм, який забезпечує миттєву синхронізацію між моделлю (даними) та представленням (інтерфейсом користувача). Завдяки цьому будь-які зміни в моделі миттєво поширюються на представлення, а будь-яка взаємодія користувача з представленням (наприклад,

введення тексту в поле) негайно оновлює модель (рис. 2.3). Це спрощує роботу з формами та взаємодію з користувачем, зменшуючи кількість необхідного коду.

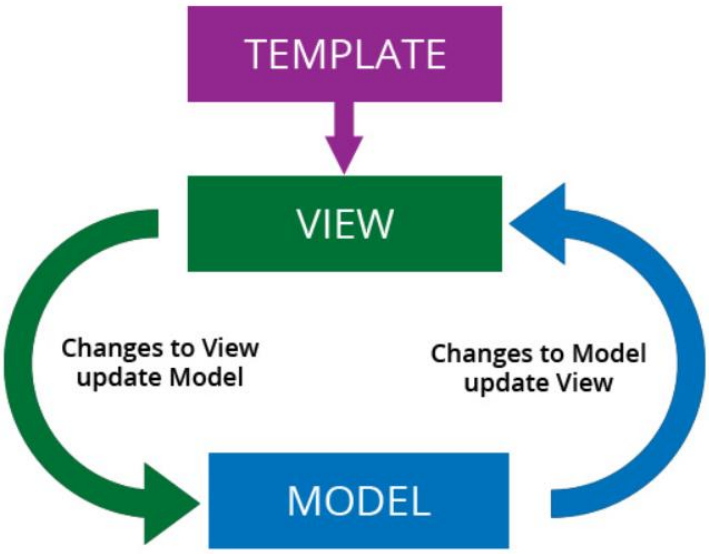

Рис. 2.3. Двостороннє зв'язування даних

Впровадження залежностей

Впровадження залежностей (Dependency Injection, DI) є процесом, за якого клас або об’єкт запитує залежності із зовнішніх джерел, а не створює їх сам. Завдяки DI можна отримати доступ до функціональності, що надається іншими класами або сервісами, не наслідуючись від них, та без необхідності знати, як ці залежності створюються або керуються. Це робить код більш гнучким і легким у підтримці.

Директиви

Директиви це – це особливі класи, що дозволяють додавати нові властивості до елементів у шаблоні або змінювати наявні. Структурні директиви змінюють DOM, додаючи або видаляючи елементи, тоді як атрибутивні директиви впливають на зовнішній вигляд або поведінку елемента, компонента чи інших директив.

Маршрутизація (роутинг)

Роутинг є механізмом, який забезпечує можливість переходу між різними представленнями або компонентами в межах одного вебзастосунку без перезавантаження всієї сторінки. Фактично, саме завдяки роутингу є можливість створювати потужні і зручні SPA-застосунки.

Інтерфейс командного рядка

Angular CLI - це інструмент командного рядка, який автоматизує рутинні задачі розробки, такі як створення проекту, додавання компонентів чи модулів, запуск тестів та збірка додатку для розгортання. [10, 11]

Розглянувши особливості Angular, можна зробити висновок, що це надзвичайно зручний інструмент для розробки вебдодатків. Але, як і будь-яка технологія, Angular має свої недоліки. Серед них можна виділити:

− великий розмір збірки, який може призвести до збільшення часу завантаження додатку, особливо на мобільних пристроях або пристроях з обмеженими мережевими ресурсами;

− складність у вивченні: На вивчення і освоєння фреймворку піде більше часу в порівнянні з аналогами, адже він потребує знання TypeScript, RxJS та має багато складних для вивчення концепцій;

− відносно невелика спільнота: Порівняно з іншими фреймворками Angular має менше активних користувачів, що може вплинути на доступність допомоги та ресурсів для вирішення проблем.

Незважаючи на недоліки Angular, саме цей фреймворк є ідеальним вибором для розробки клієнтської частини додатку для управління проектами. Він надає широкий набір можливостей, що дозволяє створювати потужні та ефективні застосунки.

2.2. Мова програмування TypeScript

TypeScript – це об’єктно-орієнтована мова програмування, розроблена Microsoft, в основі якої лежить JavaScript. Простий JavaScript-код через динамічну типізацію може призводити до неочікуваних ефектів та наслідків, особливо на великих проектах. TypeScript, в свою чергу, додає статичну типізацію, за допомогою чого є можливість визначати типи для змінних, параметрів функцій та інших об’єктів. Це полегшує виявлення помилок на етапі написання коду, а не під час його виконання, забезпечуючи більшу надійність і зручність у налагодженні.

Так як веббраузер обробляє лише JavaScript, код написаний на TypeScript потрібно скомпілювати у JavaScript перед його виконанням. Тому, після написання коду на TypeScript, він компілюється у JavaScript, який можна розгорнути в браузері або середовищі виконання Node.js (рис. 2.4.).

Рис. 2.4. Компіляція TypeScript

Серед основних особливостей TypeScript можна виділити настпуні:

− можливість визначати типи змінних, параметрів функцій та повертаємих значень;

− інтерфейси, які дозволяють визначати структуру об’єктів, що забезпечує кращу перевірку типів і підтримку інкапсуляції;

− підтримка класів, наслідування та інших концепцій ООП (Об’єктно-Орієнтоване Програмування);

− можливість організовувати код у модулі для кращої структури та підтримки проекту. [12]

2.3. Середовище виконання коду Node.js

Node.js є крос-платформним середовищем для виконання JavaScript-коду. Завдяки ньому JavaScript можна виконувати поза межами веббраузера, що дозволяє розробляти серверні додатки, використовуючи дану мову програмування, а також використовувати фреймворки, написані на JavaScript. Node.js надає засоби для роботи з файловою системою, мережевими операціями, обробкою HTTP-запитів тощо.

Базується це середовище на движку V8, який компілює JavaScript у швидкий машинний код. Крім того, Node.js має велику активну спільноту розробників, широкий вибір додаткових інструментів та бібліотек, а також підтримку модульної структури. Node.js також має власний менеджер пакетів, який полегшує установку і управління залежностями. Беручи до уваги ці фактори, дане середовище розробки було обрано для реалізації серверної частини застосунку для управління проектами.[13]

2.4. Фреймворк Express.js

Express.js є найпопулярнішим фреймворком для Node.js, який дозволяє будувати вебдодатки та API. Він доволі гнучкий, що робить його ідеальною базою для низькі інших фреймворків, таких як Sails.js та NestJS.

Express.js надає набір інструментів для обробки HTTP-запитів, відповідей та маршрутизації. Також він має потужну систему middleware, що дозволяє обробляти запити та відповіді на різних етапах їх проходження. Фреймворк відомий своєю мінімалістичністю, а отже, не нав’язує жодних структур чи шаблонів, залишаючи розробникам повну свободу у виборі архітектури додатка. Його легкість у

використанні та широка спільнота роблять його популярним вибором для створення вебдодатків різних масштабів. [13]

2.5. Система керування базами даних MongoDB

MongoDB – це СКБД, яка ґрунтується на концепції NoSQL (Non Structured Query Language). NoSQL означає, що система не використовує реляційні таблиці, як традиційні бази даних, а має документоорієнтований підхід до зберігання інформації.

Так як система має документоорієнтований підхід, то дані зберігаються у вигляді документів – структур, що складаються з пар полів і значень. Кожен запис у базі представляється у вигляді JSON-подібного (JavaScript Object Notation) об’єкта, де поля можуть містити як прості значення так і більш складні структури даних.

Великою перевагою MongoDB є гнучкість щодо схеми даних. На відміну від реляційних баз, де схема повинна бути строго визначеною, у MongoDB можна зберігати дані без попереднього визначення структури. Це дозволяє легко проводити масштабування у міру розвитку додатка, додаючи нові поля до документів без необхідності зміни всієї бази.

Щодо операцій роботи з даними, то у MongoDB доступний широкий вибір, від базових запитів на отримання, оновлення та видалення документів до складних операцій обробки даних. [14]

Використання MongoDB в контексті застосунку для управління проектами є гарним вибором, оскільки це забезпечує ефективне зберігання інформації про користувачів, їх проектів та іншої необхідної інформації.

2.6. Редактор вихідного коду Visual Studio Code

Visual Studio Code, є безкоштовним інструментом редагування коду від компанії Microsoft (рис. 2.5). Цей редактор завоював велику популярність як серед початківців у програмуванні, так і серед досвідчених розробників.

Привабливість редактора полягає в зручному та інтуїтивно зрозумілому інтерфейсі, швидкості налаштування та великому виборі різноманітних розширень, доступних для встановлення. Розширення можна завантажити з вбудованого Marketplace. Це дозволяє додавати нові функції, мови програмування, теми тощо.

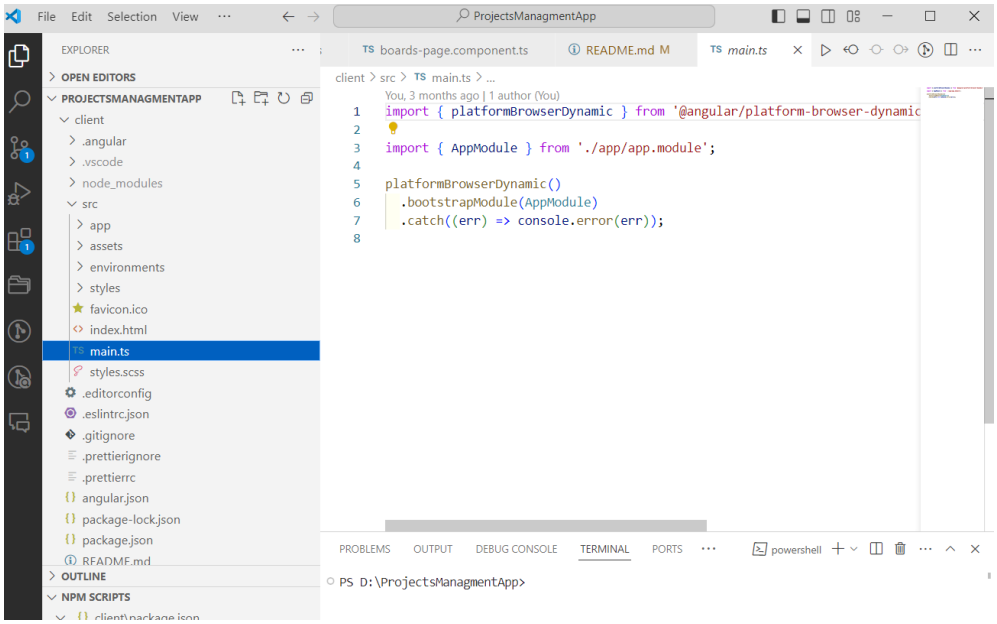

Рис. 2.5. Інтерфейс VS Code

Серед функцій VS Code варто відзначити можливість автопідстановки або ж автодоповнення, що значно полегшує і пришвидшує написання коду. Підсвітка синтаксису забезпечує кольорове виділення різних елементів коду, що покращує сприйняття структури програми. Також важливою є автоматична заміна, яка дозволяє швидко змінювати текст.

Надзвичайно корисними є вбудований термінал та підтримка системи контролю версій Git. За допомогою вбудованого терміналу можна зручно запускати сервер, відстежувати помилки та працювати з різними скриптами. Підтримка Git

дозволяє ефективно керувати версіями свого коду, робити злиття та проводити вирішення конфліктів безпосередньо у редакторі. [15]

2.7. Платформа для контейнеризації Docker

Docker – це платформа, за допомогою якої можна розробляти, доставляти та запускати програмне забезпечення в ізольованих середовищах (рис. 2.6). Docker надає можливість об’єднувати програми та всі необхідні залежності в віртуальні контейнери. Контейнери ізолюють програмне забезпечення від зовнішнього середовища, забезпечуючи постійність та незмінність. Також такі контейнери запускаються і працюють швидше за віртуальні машини, а запустити їх можна на будь-якому комп’ютері з підтримкою Docker. [16]

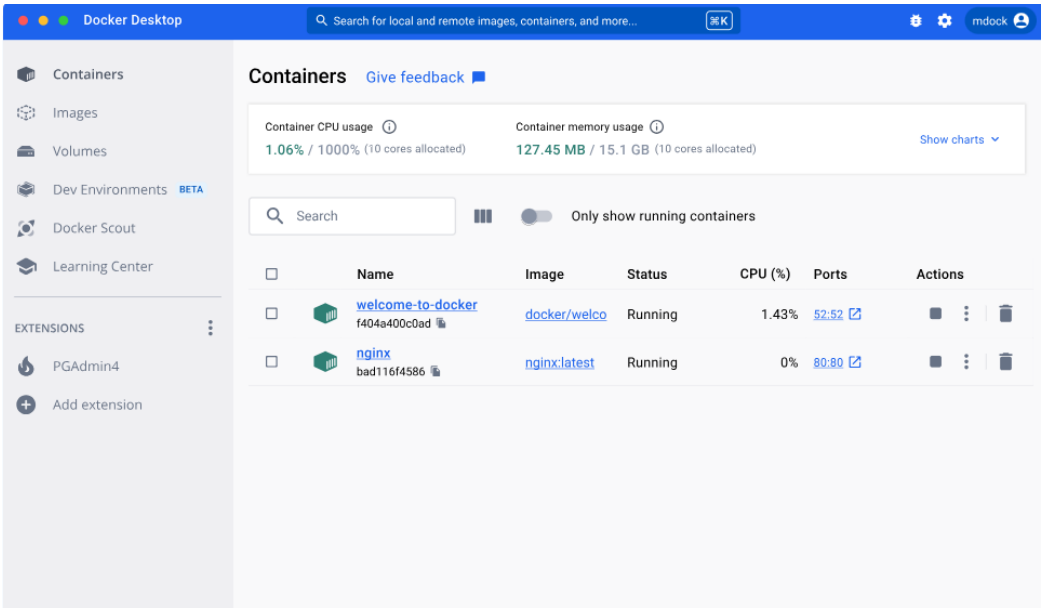

Рис. 2.6. Інтерфейс програми Docker Desktop

Наприклад, в якості СКБД застосунку, було використано MongoDB. Встановлення та налаштування даної системи може стати викликом для розробника через потребу в правильній конфігурації середовища, налаштування параметрів безпеки та управління даними.

Використання Docker у такій ситуації робить процес встановлення та налаштування MongoDB набагато простішим. Замість того, щоб вручну встановлювати та налаштовувати базу даних, можна створити Docker контейнер, який містить MongoDB та всі необхідні налаштування. За допомогою такого підходу, забезпечується простота розгортання, легкість управління базою та можливість зручного тестування додатку з різними версіями MongoDB.

2.8. Висновок до розділу 2

У другому розділі кваліфікаційної роботи було проведено огляд і аналіз технологій та інструментів, необхідних для розробки вебзастосунку для управління проектами.

Детально було розглянуто фреймворк Angular, мову програмування TypeScript, платформу для виконання коду Node.js, фреймворк Express.js, СКБД MongoDB, редактор коду Visual Studio Code та платформу для контейнеризації Docker.

Кожна з розглянутих технологій має особливості та переваги, а саме:

− Angular забезпечує великий набір функцій для створення інтерфейсів клієнтської частини;

− мова програмування TypeScript дозволяє покращити надійність та ефективність розробки як на клієнтській, так і на серверній частинах;

− середовище виконання коду Node.js і Express.js дозволяють зручно розробляти серверну логіку;

− MongoDB забезпечує гнучкість та надійне збереження даних;

− редактор коду VS Code та платформа Docker допомагають у розробці та підтримці вебдодатку;

Разом всі ці інструменти створюють потужну технічну базу для ефективної розробки вебзастосунку для управління проектами.

РОЗДІЛ 3
ВЕБЗАСТОСУНОК ДЛЯ СИСТЕМИ УПРАВЛІННЯ ПРОЕКТАМИ

3.1. Основні вимоги до вебзастосунку

Для успішного впровадження вебзастосунку для управління проектами необхідно розробити серверну та клієнтську частини додатку, а також потурбуватися про інтеграцію з базою даних. Але перш ніж переходити до безпосередньої розробки, важливо визначитись з вимогами, яким має відповідати застосунок.

До функціональних вимог належать можливість створення, редагування та видалення проектів (дошок), можливість бачити всі створені користувачем дошки та переміщувати їх у важливі, а також можливість управляти завданнями. Організаційна структура проектів буде представлена у вигляді дошок Kanban, що дозволятиме перетягувати завдання між колонками для зміни їх статусу. Також необхідно забезпечити надійну і зручну автентифікацію користувачів.

Нефункціональні вимоги включають інтуїтивно зрозумілий інтерфейс користувача, логічну та просту навігацію, високу швидкість завантаження та обробки даних, підтримку роботи з великою кількістю проектів та завдань без втрати продуктивності. Важливим аспектом під час розробки є забезпечення безпеки, що передбачає захист даних користувачів за допомогою шифрування та захист від несанкціонованого доступу. Врахування цих вимог забезпечить створення надійного, ефективного та зручного вебзастосунку для управління проектами.

3.2. Розробка серверної частини

3.2.1. Проектування та підключення бази даних

Почати роботу з базою даних варто з підключення її до додатку та налаштування середовища для роботи з нею.

Для зручності і швидкості розгортання MongoDB рекомендується використовувати Docker, так як це спрощує процес управління базою.

Для початку створимо і запустимо Docker-контейнер з зображенням MongoDB, який буде мати назву нашого додатку (рис. 3.1).

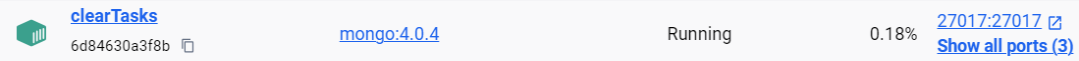

Рис. 3.1. Запущений контейнер з базою даних

Далі за допомогою Mongoose, яка є бібліотекою для MongoDB, для спрощення роботи з базою даних у Node.js, підключимо базу до проекту (рис. 3.2). Mongoose дозволяє створювати схеми для колекцій MongoDB, забезпечує валідацію даних, обробку запитів тощо. [17]

```
mongoose.connect('mongodb://localhost:27017/clearTasks').then(() => {
  console.log('connected');

  httpServer.listen(4001, () => {
    console.log("Api is listening on port 4001");
  });
})
```

Рис. 3.2. Підключення бази даних до проекту

Після підключення бази даних необхідно визначити основні сутності, якими є користувач (User), проекти (Boards), колонки (Columns) та завдання (Tasks).

Схема користувача (User)

Дана схема містить інформацію про дані користувачів такі як пошта, пароль та ім’я (рис. 3.3). На деяких полях можна побачити правила валідації, які забезпечують правильність введених даних. Це гарантує, що дані, які зберігаються, відповідають певним правилам і вимогам.

```
const userSchema = new Schema<UserDocument>({
    email: {
      type: String,
      required: [true, 'Email is required'],
      validate: [validator.isEmail, 'Invalid email'],
      createIndexes: { unique: true },
    },
    username: {
      type: String,
      required: [true, 'Username is required'],
    },
    password: {
      type: String,
      required: [true, 'Password is required'],
      select: false,
    },
  },
  {
    timestamps: true
  }
);
```

Рис. 3.3. Схема користувача у MongoDB

Схема проекту (Board)

Схема проектів містить інформацію про дошки Kanban, що використовуються для управління завданнями (рис. 3.4). Дана схема включає назву дошки, дати створення, оновлення та останнього відвідування, інформацію про те чи є дана дошка важливою, ідентифікатор фонового зображення, а також ідентифікатор користувача, який її створив.

```
const boardSchema = new Schema<BoardDocument>({
  title: {
    type: String,
    required: true
  },
  imageId: {
    type: Number,
    required: true,
    default: 0
  },
  isImportant: {
    type: Boolean,
    default: false
  },
  userId: {
    type: Schema.Types.ObjectId,
    required: true
  },
  lastJoinedAt: {
    type: Date
  }
}, { timestamps: true }
);
```

Рис. 3.4. Схема дошки у MongoDB

Схема завдань (Task)

Схема завдань включає такі поля, як назву і опис завдання, позицію завдання в колонці (для впорядкування), дати створення і оновлення, а також ідентифікатори користувача, дошки і колонки (рис. 3.5).

```
const taskSchema = new Schema<TaskDocument>({
  title: {
    type: String,
    required: true,
  },
  description: {
    type: String,
  },
  userId: {
    type: Schema.Types.ObjectId,
    required: true,
  },
  boardId: {
    type: Schema.Types.ObjectId,
    required: true,
  },
  columnId: {
    type: Schema.Types.ObjectId,
    required: true,
  },
  position: { type: Number},
}, {
  timestamps: true
});
```

Рис. 3.5. Схема завдання у MongoDB

Схема колонок (Column)

Колонки містять інформацію про завдання у межах однієї дошки. Схема колонок включає такі поля як назву колонки, ідентифікатор проекту, до якого належить колонка та ідентифікатор користувача, який її створив (рис. 3.6).

```
const columnSchema = new Schema<ColumnDocument>({
  title: {
    type: String,
    required: true,
  },
  userId: {
    type: Schema.Types.ObjectId,
    required: true,
  },
  boardId: {
    type: Schema.Types.ObjectId,
    required: true,
  },
},{
  timestamps: true
});
```

Рис. 3.6. Схема колонки у MongoDB

Після визначення основних сутностей можна починати працювати з базою даних та обробляти дані в залежності від потреб.

3.2.2. Розробка API для взаємодії клієнтської частини з сервером

В контексті вебзсатосунку для управління проектами API буде складатися з двох частин, а саме:

– HTTP API, що дозволить клієнту виконувати CRUD операції з даними за допомогою стандартних HTTP-запитів.

– WebSocket API, що забезпечить двонаправлений зв'язок між клієнтом та сервером у реальному часі. Це дозволить прововодити швидке оновлення даних без перезавантаження сторінки.

Для реалізації API перш за все, необхідно створити і запустити HTTP-сервер за допомогою Express.js (рис. 3.7).

```
21  const app = express();
22  const httpServer = createServer(app);
23
24  app.use(cors());
25  app.use(bodyParser.json())
26  app.use(bodyParser.urlencoded({extended: true}))
27
```

Рис. 3.7. Налаштування серверу за допомогою Express.js

Далі створимо WebSocket-сервер за допомогою бібліотеки socket.io, яка забезпечує просту інтеграцію WebSocket у додатки (рис. 3.8).

```
29  const io = new Server(httpServer, {
30    cors: {
31      origin: "*",
32    },
33  });
34
35  io.on('connection', () => {
36    console.log('connected');
37  })
```

Рис. 3.8. Налаштування серверу за допомогою socket.io

Після налаштування серверів, можна переходити до розробки маршрутів та подій для обробки запитів і взаємодії з базою даних. Маршрути визначають кінцеві точки (endpoints), за якими клієнтська частина може звертатися до сервера.

Спочатку обробимо події для WebSocket-з'єднання, за допомогою яких буде здійснюватися управління дошками, колонками і завданнями у реальному часі (рис. 3.9). Кожна подія викликає відповідний метод контролера, який обробляє запит і виконує необхідні дії.

```
io.use(async (socket: Socket, next) => {      You, 2 weeks ago
}).on("connection", (socket) => {
  socket.on(SocketEventsEnum.boardsJoin, (data) => {
    boardsController.joinBoard(io, socket, data);
  });
  socket.on(SocketEventsEnum.boardsLeave, (data) => {
    boardsController.leaveBoard(io, socket, data);
  });
  socket.on(SocketEventsEnum.columnsCreate, (data) => {
    columnsController.createColumn(io, socket, data);
  });
  socket.on(SocketEventsEnum.tasksCreate, (data) => {
    tasksController.createTask(io, socket, data);
  });
  socket.on(SocketEventsEnum.boardsUpdate, (data) => {
    boardsController.updateBoard(io, socket, data);
  });
  socket.on(SocketEventsEnum.boardsDelete, (data) => {
    boardsController.deleteBoard(io, socket, data);
  });
  socket.on(SocketEventsEnum.columnsDelete, (data) => {
    columnsController.deleteColumn(io, socket, data);
  });
  socket.on(SocketEventsEnum.columnsUpdate, (data) => {
    columnsController.updateColumn(io, socket, data);
  });
  socket.on(SocketEventsEnum.tasksUpdate, (data) => {
    tasksController.updateTask(io, socket, data);
  });
  socket.on(SocketEventsEnum.tasksDelete, (data) => {
    tasksController.deleteTask(io, socket, data);
  });
});
```

Рис. 3.9. Обробка WebSocket-подій

Далі обробимо маршрути для HTTP-запитів, які забезпечують реєстрацію автентифікацію, отримання інформації про користувача, дошки, колонки та

завдання (рис. 3.10).

```
app.post('/api/users/register', usersController.register);
app.post('/api/users/login', usersController.login);
app.get('/api/user', authMiddleware, usersController.currentUser);
app.get('/api/boards', authMiddleware, boardsController.getBoards);
app.post('/api/boards', authMiddleware, boardsController.cretateBoard);
app.get("/api/boards/:boardId", authMiddleware, boardsController.getBoard);
app.get(
  "/api/boards/:boardId/columns",
  authMiddleware,
  columnsController.getColumns
);
app.get("/api/boards/:boardId/tasks", authMiddleware, tasksController.getTasks);
app.patch("/api/boards/:boardId", authMiddleware, boardsController.updateBoardSimple);
```

Рис. 3.10. Обробка маршрутів для HTTP-запитів

При зверненні до окремого маршруту сервер виконує відповідний контролер для обробки запиту. Наприклад, маршрут app.get(‘/api/boards’, authMiddleware, boardsController.getBoards); обробляє GET-запити на /api/boards. Він використовує проміжний обробник authMiddleware для перевірки чи є користувач автентифікованим. Якщо це так, то викликається метод getBoards контролера boardsController (рис. 3.11).

```
export const getBoards = async (
  req: ExpressRequestInterface,
  res: Response,
  next: NextFunction
) => {
  try {
    if (!req.user) {
      return res.status(401);
    }
    const boards = await BoardModel.find({userId: req.user.id });
    res.send(boards);
  } catch (err) {
    next(err);
  }
}
```

Рис. 3.11. Реалізація методу getBoards

Для кожної події та кожного маршруту було створено окремий контролер. Це забезпечує кращу організацію коду, полегшує його підтримку та розширення.

Кожен контролер відповідає за конкретну дію, що дозволяє чітко розмежувати функціональність і забезпечити зрозумілу архітектуру застосунку.

Отже, в результаті даного етапу розробки було реалізовано повноцінну, готову до використання серверну частину вебзастосунку для управління проектами.

Також важливо зазначити, що при розробці було використано саме TypeScript, а не звичайний JavaScript. Це дозволило додати статичну типізацію до коду, що допомогло зменшити кількість помилок і підвищити надійність.

3.3. Розробка клієнтської частини

Розробка клієнтської частини вебзастосунку для управління проектами є ключовим етапом, що забезпечує зручний і інтуїтивно зрозумілий інтерфейс для взаємодії з користувачами. Реалізована вона з використанням фреймворку Angular.

Перш за все, необхідно ініціалізувати проект. Зручніше за все це зробити за допомогою Angular CLI, що забезпечує швидке створення проекту зі стандартною структурою (рис. 3.12).

```
ng new projectsManagmentApp
cd projectsManagmentApp
```

Рис. 3.12. Ініціалізація нового проекту за допомогою Angular CLI

За допомогою Angular CLI було створено проект, а також додано всі необхідні бібліотеки. Однак, додатково необхідно встановити деякі додаткові залежності, такі як socket.io-client для роботи з WebSocket зі сторони клієнту та @angular/material для покращення зовнішнього вигляду і зручності в розробці користувацького інтерфейсу (рис. 3.13).

```
npm install @angular/material @angular/cdk @angular/animations
npm install socket.io-client
```

Рис. 3.13. Завантаження пакетів socket.io-client та @angular/material

Після даного етапу необхідно визначитися зі структурою проекту. Він поділений на кілька папок, кожна з яких відповідає за певний функціонал і містить у собі відповідні компоненти та сервіси:

- auth: авторизація та автентифікація користувачів;
- boards: робота з дошками проектів;
- home: відображення домашньої сторінки;
- shared: загальні компоненти та сервіси, які використовуються в різних частинах застосунку.

Також головна папка проекту містить кілька важливих файлів і директорій:

- app-routing.module.ts: файл для налаштування маршрутизації у проекті;
- app.component.html: основний HTML-шаблон застосунку;
- app.component.ts: основний компонент застосунку;
- app.module.ts: основний модуль застосунку;
- assets/: директорія для зберігання статичних ресурсів, таких як зображення та файли;
- environments/: директорія для зберігання конфігураційних файлів для різних середовищ;
- styles/: директорія для зберігання стилів, що використовуються у проекті.

Файлова структура проекту наведена на рис. 3.14.

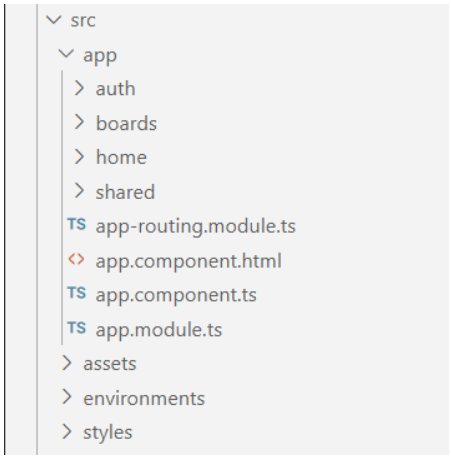

Рис. 3.14. Файлова структура проекту

Далі, можна переходити до безпосередньої розробки вебзастосунку. Почати варто з маршрутизації, що забезпечить перехід між різними частинами застосунку без перезавантаження сторінки, яку можна налаштувати за допомогою Angular Router. Для застосунку з управління проектами буде налаштовано маршрути для таких сторінок, як домашня, сторінки автентифікації, сторінка з дошками, сторінка окремої дошки, та сторінка завдання.

Наприклад, налаштуємо роутинг між сторінками входу в систему та реєстрації користувача (рис. 3.15).

```
export const AUTH_ROUTES: Routes = [
  {
    path: 'login',
    component: LoginComponent
  },
  {
    path: 'register',
    component: RegisterComponent
  }
];
```

Рис. 3.15. Налаштування маршрутизації для сторінок логіну та реєстрації

В даному випадку, можна побачити масив об’єктів, де кожен елемент обов’язково має мати такі поля як path, яке вказує на маршрут, що буде оброблятися, та component, яке визначає компонент, який буде відображатися при переході на відповідний маршрут.

За схожим принципом, було налаштовано інші маршрути додатку.

Тепер можна перейти, до реалізації сторінок, тобто головних компонентів вебзастосунку для управління проектами.

Почнемо з домашньої сторінки, яка є першим, що бачить користувач. На ній зобразимо базову інформацію про застосунок, а також надамо користувачу кнопки для навігації на сторінки реєстрації та входу (рис. 3.16).

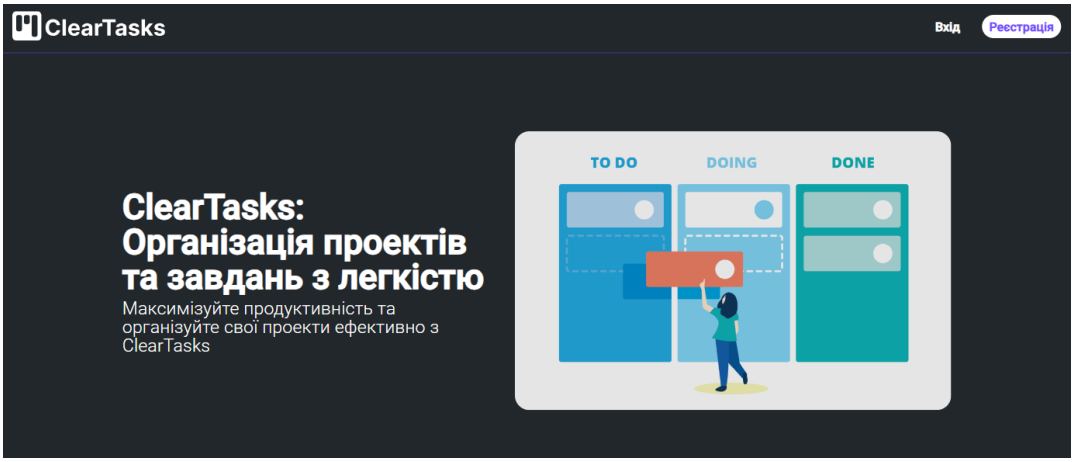

Рис. 3.16. Домашня сторінка вебзастосунку

Після впровадження домашньої сторінки, треба попіклуватися про доступ користувача до системи, а саме про сторінки для входу та реєстрації.

Для цього треба забезпечити користувача безпечним і зручним способом для здійснення автентифікації, тому треба розробити форму для вводу даних. Очевидним є те, що поля в такій форми будуть повторюватися не один раз на різних сторінках. А отже гарним рішенням буде створити окремий InputComponent для поля форми, який буде зручно перевикористовувати (рис. 3.17).

```
export class InputComponent<T> extends ControlValueAccesorDirective<T> {
  @Input() inputId = '';
  @Input() label = '';
  @Input() placeholder = '';
  @Input() type: InputType = 'text';
  @Input() formControlName = '';
  @Input() customErrorMessages: Record<string, string> = {};
  @Input() toggleIconsUrl: string[] = [];

  protected currentIconIndex = 0;

  togglePassVisibility(): void {
    this.type = this.type === 'text' ? 'password' : 'text';
    this.currentIconIndex = this.currentIconIndex === 0 ? 1 : 0;
  }
}
```

Рис. 3.17. Компонент поля форми

Такий компонент приймає в себе різноманітні значення, що дозволяє гнучко його налаштовувати відповідно до функціональних потреб. Шаблон компоненту наведено на рис. 3.18.

```
<div class="form-control" [class.invalid]="isInValid()">
  <label *ngIf="label" [for]="inputId" [class.required]="isRequired">{{ label }}</label>
  <div class="input-container">
    <input
      [required]="isRequired"
      [type]="type"
      [id]="inputId"
      [formControl]="$any(control)"
      [placeholder]="placeholder"
      autocomplete="on"
    />
    <ng-container *ngIf="toggleIconsUrl && toggleIconsUrl.length > 0">
      <img
        class="input-icon"
        (click)="togglePassVisibility()"
        [src]="toggleIconsUrl[currentIconIndex]"
        alt=""
      />
    </ng-container>
  </div>
  <app-validation-errors
    *ngIf="isInValid()"
    [errors]="control!.errors"
    [controlName]="formControlName"
    [control]="$any(control)"
  >
  </app-validation-errors>
</div>
```

Рис. 3.18. Шаблон компонент поля форми

Тепер завдяки такому елементу, можна зручно і швидко розробляти користувацькі форми.

Наступним кроком буде реалізація форми для користувача, який ще не зареєстрований у системі (рис. 3.19).

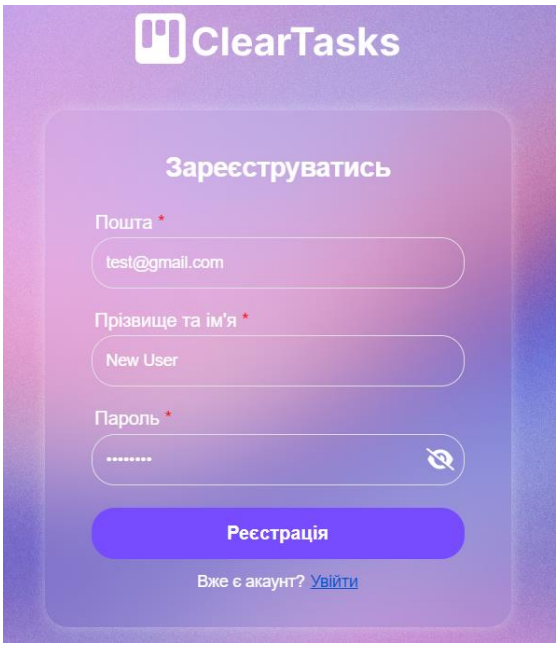

Рис. 3.19. Форма на сторінці реєстрації

Якщо користувач вже має обліковий запис, він зможе увійти в систему, увівши необхідні дані на відповідній сторінці для входу. Після успішного входу у застосунок, його буде переправлено на головну сторінку додатку, яка водночас є сторінкою на якій зображено усі дошки створені користувачем (рис. 3.20).

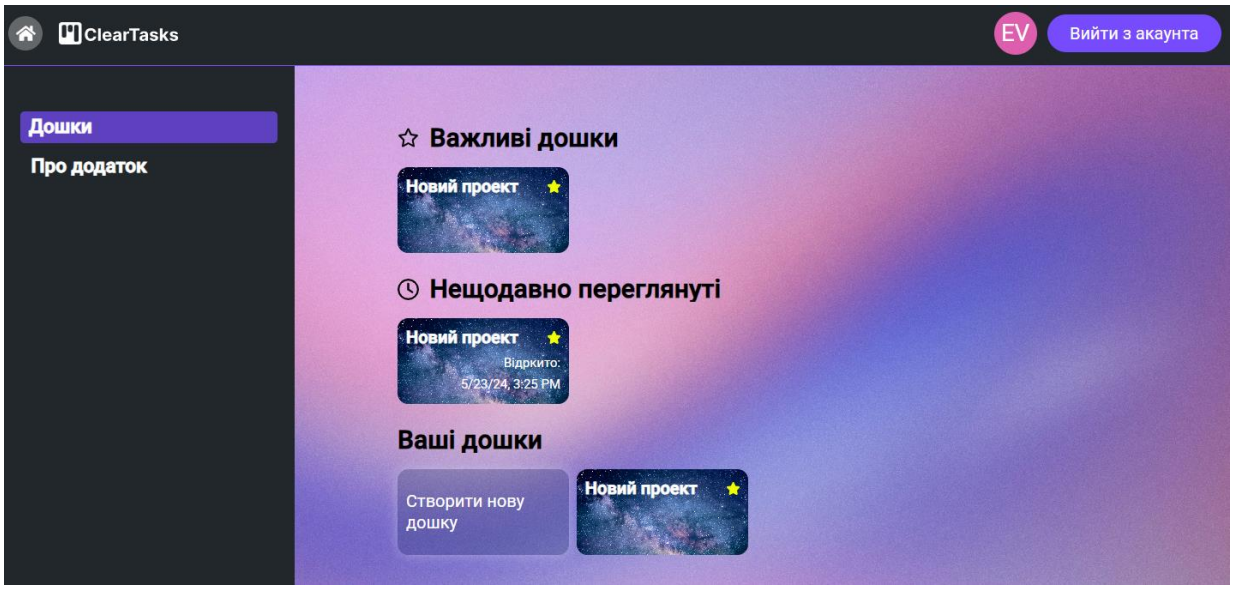

Рис. 3.20. Головна сторінка вебзастосунку

Дану сторінку можна розбити на декілька ключових частин, а саме бокову панель навігації, поле з дошками та верхню панель. На верхній панелі присутні елементи навігації, а також кнопка для завершення користувацької сесії. При натисканні на цю кнопку викликається метод logout(), який завершує сесію користувача та перенаправляє його на домашню сторінку (рис. 3.21).

```
protected authService = inject(AuthApiService);
private router = inject(Router);

logout(): void {
  this.authService.logout();
  this.router.navigateByUrl('/');
}
```

Рис. 3.21. Реалізація методу logout()

На полі з дошками знаходиться кнопка, за допомогою якої є можливість створити новий проект.

При натисканні на дану кнопку відкривається модальне вікно, яке містить поле для вводу назви дошки, та скролбар з різноманітними варіантами фонового зображення, яке буде застосовано до неї (рис. 3.22). Як тільки користувач увів назву, кнопка для створення дошки стає активною і на неї можна натиснути. Після натискання, посилається запит на сервер і після його обробки нова дошка з’являється у списку (рис. 3.23).

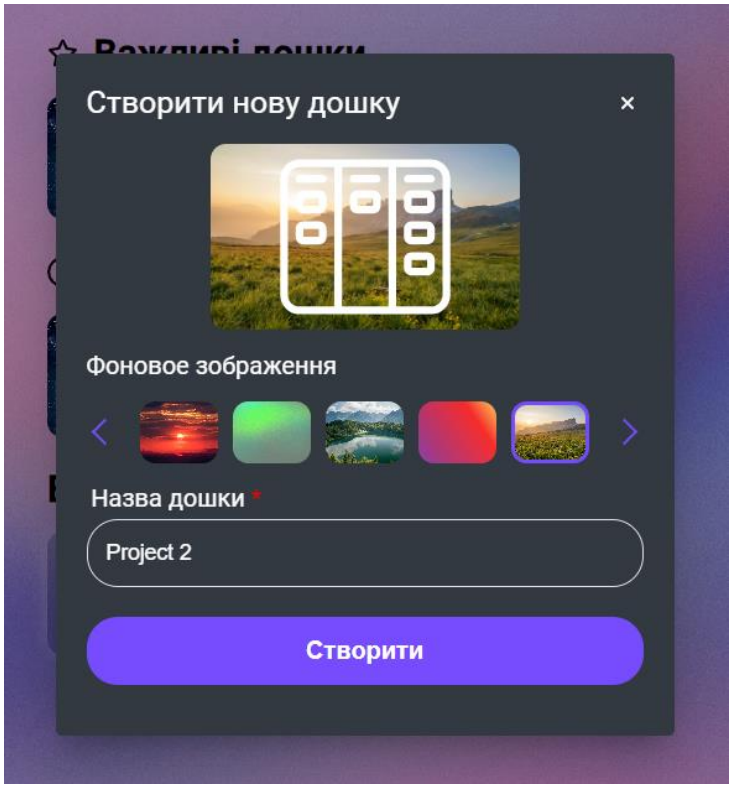

Рис. 3.22. Модальне вікно для створення дошки

Рис. 3.23. Щойно створена дошка

Після того, як дошка була створена, можна перейти на неї і почати працювати над створенням завдань. Кожен проект складається з колонок, які всередині містять картки завдань. Користувач може створити безліч колонок з різноманітними назвами на свій смак, але зазвичай вони відображають стадії виконання завдань, такі як “ToDo”, “In Progress”, “Done” тощо (рис. 3.24).

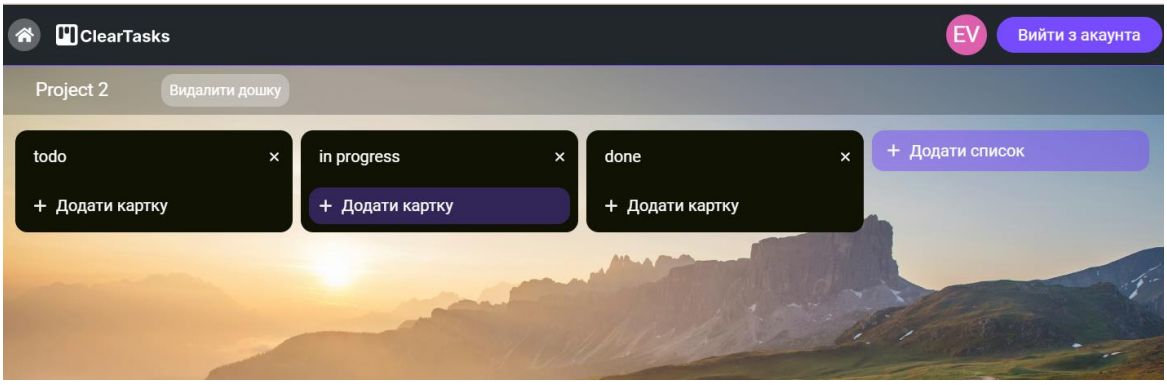

Рис. 3.24. Дошка з пустими колонками

Тепер, необхідно створити картки завдань. Завдання можна створювати в межах кожної колонки, що дозволяє користувачам чітко бачити, на якому етапі виконання знаходиться кожна задача. При натисканні на “Додати картку” з’являється поле в яке треба ввести назву завдання, після чого треба натиснути на кнопку, яка обробить подію створення нової картки. На рис. 3.25 наведений метод onCreateTaskClick() обробки такої події. Він ініціює процес створення завдання через сервіс, а саме метод createTask(), який взаємодіє із сервером та забезпечує оновлення в реальному часі через Socket.io (рис. 3.26).

```
onCreateTaskClick(title: string, columnId: string, position: number): void {
  const taskInput: TaskInputInterface = {
    title,
    boardId: this.boardId,
    columnId,
    position
  };

  this.tasksService.createTask(taskInput);
}
```

Рис. 3.25. Метод onCreateTaskClick() для створення нової картки завдань

```
createTask(taskInput: TaskInputInterface): void {
  this.socketService.emit(SocketEventsEnum.tasksCreate, taskInput);
}
```

Рис. 3.26. Метод сервісу createTask() для взаємодії з Socket.io

За допомогою функціональності Drag and Drop (перетягування) можна легко переміщувати картки завдань між колонками. Це дозволяє чітко відстежувати статус завдань. При натискання на картку завдання, відкривається модальне вікно (рис. 3.27). За допомогою модального вікна і елементів у ньому, можна перемістити завдання у іншу колонку, додати до нього опис та навіть видалити картку.

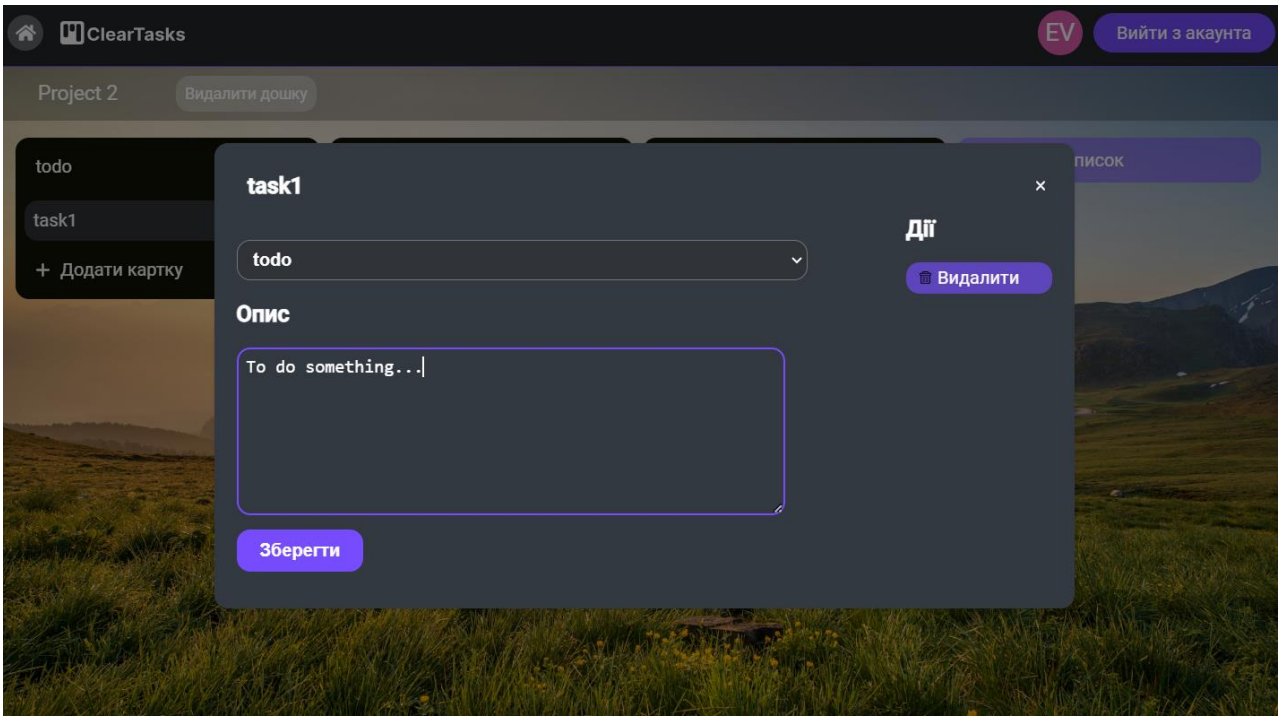

Рис. 3.27. Модальне вікно редагування завдання

Отже, тепер можна повноцінно працювати з дошкою, створюючи безліч завдань, переміщуючи та редагуючи їх через модальне вікно.

Приклад повністю функціонуючої дошки наведено на рис. 3.28. Користувач також має змогу редагувати назву проекту та видаляти його за необхідності.

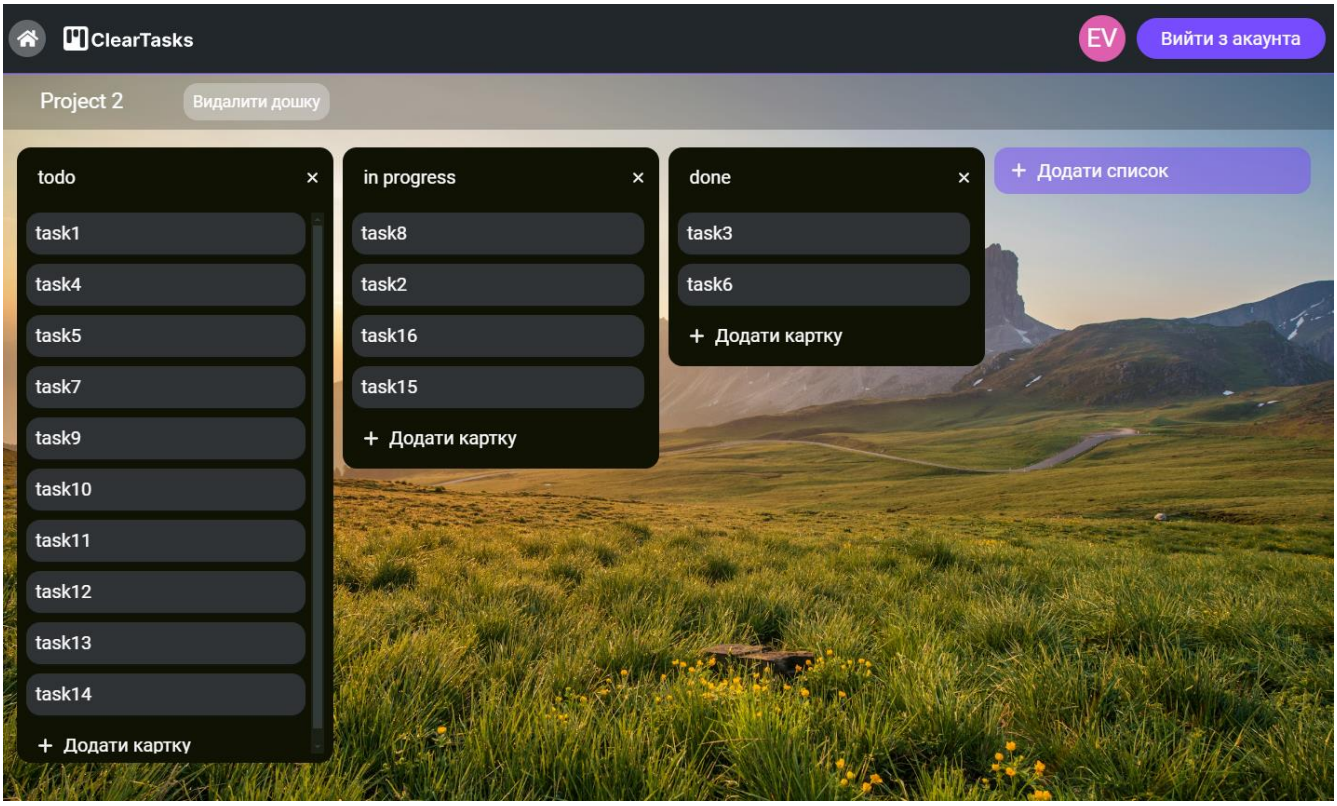

Рис. 3.28. Повноцінна працююча дошка

3.4. Висновок до розділу 3

У третьому розділі було проведено реалізацію вебзастосунку для управління проектами.

Розглянуто та визначено основні вимоги до вебзастосунку, які включали як функціональні, так і нефункціональні вимоги.

Розроблено серверну частину застосунку, що включало в себе створення та підключення бази даних, а також розробку API для взаємодії клієнтської частини з сервером. Було показано головні етапи розробки та створено основні сутності бази даних. Крім того, було розглянуто створення API, яке дозволило клієнтській частині взаємодіяти з сервером.

Описано процес розробки клієнтської частини вебзастосунку. Було налаштовано маршрутизацію для забезпечення переходів між різними сторінками,

реалізовано основні сторінки та додаткові компоненти. Було також реалізовано Drag and Drop функціональність для зручного переміщення завдань між колонками.

В даному розділі було створено повноцінний вебзастосунок, який забезпечує всі необхідні функції для ефективного планування, виконання та моніторингу завдань. Використання сучасних технологій, таких як Angular, Node.js, MongoDB тощо, дозволило створити надійний та зручний у використанні інструмент для управління проектами

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було реалізовано комплексну систему управління проектами, з використанням сучасних технологій веброзробки. Завдяки реалізації даної системи було підвищено ефективність управління завданнями шляхом створення зручного та функціонального вебзастосунку. Такий застосунок в значній мірі спростить процеси планування, виконання та контролю проектів.

Під час виконання роботи було розглянуто основні поняття веброзробки, зокрема переваги і недоліки клієнт-серверної архітектури, протоколи HTTP та WebSocket. Клієнт-серверна архітектура дозволяє чітко розподілити навантаження між клієнтом і сервером, що підвищує масштабованість і надійність системи, а протоколи HTTP та WebSocket забезпечують ефективний та зручний обмін даними.

Було проведено огляд існуючих рішень для управління проектами, таких як JIRA, Monday.com та Miro, що дозволило виявити їхні сильні та слабкі сторони, а також проаналізувати вимоги до розробки нового застосунку. Виявлені недоліки в існуючих рішеннях допомогли визначити ключові аспекти, які треба було врахувати при розробці власного вебзастосунку, такі як інтуїтивно зрозумілий інтерфейс, набір необхідних функцій та інструментів для ефективного управління проектами, доступність та стабільність навіть при високому навантаженні.

Було також здійснено огляд технологій та інструментів, необхідних для розробки. Особливу увагу було приділено фреймворку Angular, який забезпечив структурованість та ефективність розробки клієнтської частини. Angular надає потужний набір інструментів для створення вебзастосунків, що дозволяє зменшити кількість помилок та підвищити надійність коду. Використання TypeScript як основної мови програмування додало статичну типізацію, що підвищило стабільність і передбачуваність роботи додатку.

Процес впровадження вебзастосунку включав розробку серверної та клієнтської частин. Було здійснено проектування бази даних та розроблено API. Це дозволило створити надійну інфраструктуру для обробки та зберігання даних. Після цього було забезпечено зручний користувацький інтерфейс, який відзначається приємним для ока дизайном та швидкою навігацією, що робить роботу із застосунком інтуїтивно зрозумілою та комфортною.

Щодо функцій з управління проектами, то користувач був забезпечений такими можливостями, як створення, редагування та видалення дошок, колонок та завдань. Також було реалізовано функціональність для зручного переміщення завдань між колонками, що дозволяє чітко відстежувати їх статус. Крім того, додаток підтримує модальні вікна для детального редагування завдань, забезпечуючи можливість зміни назви, опису та інших параметрів.

Вебзастосунок був повністю відлагоджений, було усунено всі виявлені помилки та проблеми, а отже тепер він повністю відповідає як функціональним так і не функціональним вимогам.

Застосунок має потенціал для подальшого розвитку та покращення. Наприклад, серед можливих напрямків, може бути додавання можливості інтеграції з популярними інструментами, такими як календарі або ж електронна пошта. Також можна додати нові функції, такі як аналітика, відстеження часу, звітність тощо.

Значення такого вебзастосунку в житті полягає в його здатності значно підвищити ефективність і продуктивність роботи. Він дозволяє краще організовувати робочі процеси, що зменшує витрати часу та ресурсів. Застосунок забезпечує централізоване місце для зберігання та управління всіма проектами та завданнями.

Загалом, виконання кваліфікаційної роботи не тільки дозволило створити корисний інструмент, але й сприяло професійному розвитку та вдосконаленню навичок у сфері веброзробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is the difference between web page, website, web server, and search engine? [Electronic resource]. Access mode: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/Pages_sites_servers_and_search_engines (last access 09.05.2024). – Title from the screen.

2. Difference Between Web application and Website [Electronic resource]. Access mode: https://www.geeksforgeeks.org/difference-between-web-application-and-website/ (last access 09.05.2024). – Title from the screen.

3. Single Page Application (SPA) definition [Electronic resource]. Access mode: https://www.sanity.io/glossary/single-page-application (last access 11.05.2024). – Title from the screen.

4. Mbuguah S., Mony V., Nyabuto G. Architectural Review of Client-Server Models. *International Journal of Scientific Research and Engineering Trends*. 2024. Volume 10. P.139 – 142.

5. What is Client-Server Architecture? Explained in Detail [Electronic resource]. Access mode: https://www.theknowledgeacademy.com/blog/client-server-architecture/ (last access 12.05.2024). – Title from the screen.

6. Comparing WebSockets and HTTP [Electronic resource]. Access mode: https://websocket.org/guides/road-to-websockets/#comparing-websockets-and-http (last access 12.05.2024). – Title from the screen.

7. What is JIRA Software? [Electronic resource]. Access mode: https://www.geeksforgeeks.org/what-is-jira-software/ (last access 14.05.2024). – Title from the screen.

8. An Introduction to monday.com: What Is It and How Does It Work? [Electronic resource]. Access mode: https://www.findmycrm.com/blog/what-is-monday (last access 14.05.2024). – Title from the screen.

9. What is Miro and How to Use Miro for Virtual Collaboration [Electronic resource]. Access mode: https://www.innovationtraining.org/what-is-miro-and-how-to-use-miro-for-collaboration/ (last access 14.05.2024). – Title from the screen.

10. Freeman A. Pro Angular: Build Powerful and Dynamic Web Apps. Apress; 5th ed. 2022. 905 p.

11. What is Angular? A Beginners Guide [Electronic resource]. Access mode: https://www.theknowledgeacademy.com/blog/what-is-angular/ (last access 18.05.2024). – Title from the screen.

12. Freeman A. Essential TypeScript 4: From Beginner to Pro. Apress; 2nd ed. Edition. 2021. 581 p.

13. Brown E. Web Development with Node and Express: Leveraging the JavaScript Stack. O'Reilly Media; 2nd edition. 2019. 343 p.

14. Why Use MongoDB and When to Use It? [Electronic resource]. Access mode: https://www.mongodb.com/resources/products/fundamentals/why-use-mongodb (last access 16.05.2024). – Title from the screen.

15. Visual Studio Code [Electronic resource]. Access mode: https://www.educba.com/what-is-visual-studio-code/ (last access 16.05.2024). – Title from the screen.

16. Docker overview [Electronic resource]. Access mode: https://docs.docker.com/get-started/overview/ (last access 17.05.2024). – Title from the screen.

17. Getting Started with MongoDB & Mongoose [Electronic resource]. Access mode: https://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-and-mongoose/ (last access 20.05.2024). – Title from the screen.

18. Положення про кваліфікаційні роботи (проєкти) здобувачів вищої освіти Національного Авіаційного Університету. СМЯ НАУ П 03.01(10) – 03 – 2024. Київ 2024, 62с.