

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ ВЕБЗАСТОСУНКУ ДЛЯ
АВТОМАТИЗАЦІЇ РОБОТИ МІЖ ЗАМОВНИКОМ
І РОЗРОБНИКОМ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
DEVELOPMENT OF THE WEB APPLICATION SERVER SIDE TO
AUTOMATE THE WORK BETWEEN A CUSTOMER
AND A SOFTWARE DEVELOPER

Виконав: студент(ка) 4 курсу, групи КНТ-110сп
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

УКРАЇНЦЕВА Є. С.

(ПРИЗВИЩЕ та ініціали)

Керівник КАПЛІЄНКО Т. І.

(ПРИЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2023
 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

УКРАЇНЦЕВОЇ Євгенії Сергіївни

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка серверної частини вебзастосунку для автоматизації роботи між замовником і розробником програмного забезпечення

керівник проєкту (роботи) к.т.н., доцент, КАПЛІЄНКО Тетяна Ігорівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 04 ” квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 29 травня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	КАПЛІЄНКО Т. І., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання « 06 » лютого 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис)

Євгенія УКРАЇНЦЕВА

(Ім'я Прізвище)

Керівник проєкту (роботи)

(підпис)

Тетяна КАПЛІЄНКО

(Ім'я Прізвище)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра містить 88 с., 14 рис., 19 табл., 2 дод., 30 джерел.

ВЕБЗАСТОСУНОК, СЕРВЕРНА ЧАСТИНА, NODE JS, MONGO DB, БАЗА ДАНИХ, РОЗРОБНИК, КЛІЄНТ, ПЛАНУВАННЯ РОЗРОБКИ.

Об'єкт дослідження – програмні засоби для автоматизації взаємодії між замовником і розробником програмного забезпечення.

Предмет дослідження – серверна частина вебзастосунку для автоматизації та покращення ефективності взаємодії між замовником і розробником програмного забезпечення.

Мета роботи – дослідження розвитку сучасних серверних вебтехнологій та їх ефективного застосування на прикладі створення серверної частини вебсайту, призначеного для автоматизації роботи з проєктом під час його створення і реалізації між замовником і розробником програмного забезпечення.

Матеріали, методи та технічні засоби: структурне та компонентно-орієнтоване програмування, мова програмування JavaScript з використанням фреймворку node.js, персональний комп'ютер з процесором Intel(R) Core(TM) i5-6200U під управлінням операційної системи Ubuntu 22.04.1 LTS.

Результати. Створено серверну частину вебзастосунку, яка є складовою частиною застосування, призначеного для покращення взаємодії між замовником і розробником програмного забезпечення.

Висновки. Розроблено програмну систему для прискорення та автоматизації взаємодії розробників з клієнтами.

Галузь використання – налагодження ефективної співпраці між замовником і розробником програмного забезпечення в організаціях, що займаються розробкою програмних продуктів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 88 pages, 14 figures, 19 tables, 2 appendixes, 30 sources.

WEB APPLICATION, SERVER-SIDE, NODE JS, MONGO DB, DATABASE, DEVELOPER, CLIENT, DEVELOPMENT PLANNING.

Research object - software tools for automating interaction between the client and software developer.

Research subject - a web application for automating and improving the efficiency of interaction between the client and software developer.

The purpose of the project is to investigate the development of modern server-side web technologies and their effective application through the creation of the server-side component of a website, designed to automate project management during its creation and implementation between the client and software developer.

Materials, Methods, and Technical Tools: structural and component-based programming, JavaScript programming language with the use of node.js framework, personal computer with Intel(R) Core(TM) i5-6200U processor running Ubuntu 22.04.1 LTS operating system.

Results. The server-side component of the web application has been created, which is an integral part of an application designed to improve the interaction between the client and software developer.

Conclusions. A software system has been developed to accelerate and automate the interaction between developers and clients.

Field of application - establishing effective collaboration between software clients and developers in organizations involved in software development.

ЗМІСТ

ВСТУП	8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	9
1.1 Основні поняття	9
1.2 Огляд програмних аналогів	11
1.3 Висновки за розділом 1	15
2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ	17
2.1 Вибір засобів проєктування і розробки	17
2.2 Визначення функціональних вимог до програмного забезпечення.....	22
2.3 Проєктування бази даних.....	24
2.4 Визначення структури програми.....	28
2.5 Висновки за розділом 2	30
3 РОЗРОБКА ПРОГРАМИ.....	31
3.1 Алгоритм функціонування програми	31
3.2 Опис реалізованих складових вебзастосунку	31
3.3 Висновки за розділом 3	40
4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ	41
4.1 Опис програми	41
4.2 Експлуатація програми.....	49
4.3 Тестування програми.....	51
4.4 Висновки за розділом 4	53
ВИСНОВКИ.....	54
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	56
ДОДАТОК А Текст програми.....	59
ДОДАТОК Б Слайди презентації	80

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- API – Application Programming Interface;
HTTP – HyperText Transfer Protocol;
IDE – Integrated Development Environment;
REST – Representational State Transfer;
ПЗ – програмне забезпечення.

ВСТУП

У сучасному світі розвиток технологій дає можливість проводити глобальну автоматизацію різних сфер суспільного життя. Саме тому збільшується обсяг даних, що зберігається.

Бази даних стають все більш громіздкими та менш зручними в експлуатації. Для пошуку необхідної інформації потребується все більше часу, а потреба в ній з'являється у все більшій кількості людей, які знаходяться на різних відстанях.

За таких умов стає необхідним знайти рішення для ефективного та швидкого використання великих обсягів даних, а також для доступу до цих ресурсів незалежно від місця знаходження користувачів. Таким рішенням стало Інтернет. Сьогодні Інтернет є найбільш розвиненою інформаційною системою в світі, яка забезпечує комунікацію між мільйонами користувачів та їх інформаційні потреби. Для зручного обміну інформацією використовуються вебтехнології, які постійно надають нові можливості.

Метою даної дипломної роботи є дослідження розвитку сучасних вебтехнологій та їх ефективного застосування на прикладі створення вебсайту, призначеного для автоматизації роботи з проєктом під час його створення і реалізації між замовником і розробником програмного забезпечення.

На сьогодні представлення організацій у мережі Інтернет є необхідним для покращення ефективності роботи організації, обміну інформацією між усіма учасниками ринкового процесу. Саме тому темою дипломної роботи було обрано створення ПЗ для автоматизації роботи між замовником і розробником програмного забезпечення.

Серверну частину вебзастосунку було розроблено за допомогою середовища програмування Visual Code з використанням фреймворку Node.js.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні поняття

Предметна область визначається як сфера людської діяльності, яка відповідає певним критеріям і описується згідно з ними.

У цьому контексті термін «модель» відноситься до будь-якого математичного об'єкта, який описує властивості досліджуваної системи.

Модель предметної області виконує роль системи, яка імітує структуру або функціонування самої предметної області і має бути відповідною до неї.

Кожен аспект предметної області складається з безлічі об'єктів і процесів, які взаємодіють між собою, а також з різних користувачів, які мають різні погляди на предметну область. У даній роботі було поставлено завдання створити серверну частину програмного забезпечення для автоматизації роботи між замовником і розробником програмного забезпечення. Тому під час виконання роботи було проведено дослідження цієї предметної області.

Часто замовники програмного забезпечення не мають можливості швидко зв'язатися з розробниками для коригування деталей проєкту. Вони не завжди чітко бачать та розуміють етапи розробки. Це може ускладнювати процес розробки програмного забезпечення. В той же час, фахівці з розробки забезпечення потребують чіткого розуміння сутності, мети, етапів та часових рамок проєкту.

Темою дипломної роботи було обрано створення серверної частини програмного забезпечення для автоматизації роботи між замовником і розробником.

Процес розробки програмного забезпечення є послідовністю дій, спрямованих на створення та підтримку працездатності, якості та надійності програмного забезпечення. Він використовує технології, методологію та

практики з інформатики, керування проєктами, математики, інженерії та інших областей знань.

Процес розробки складається з безлічі підпроцесів, або дисциплін, деякі з яких показані нижче:

- аналіз вимог;
- проєктування програмного забезпечення;
- програмування;
- тестування програмного забезпечення;
- системна інтеграція;
- впровадження програмного забезпечення;
- супровід програмного забезпечення [1].

Розробник програмного забезпечення — фахівець, який займається безпосередньою розробкою програмного забезпечення для різного роду обчислювально-операційних систем.

Програмне забезпечення — це загальний термін для різних видів програм, що використовуються для роботи з комп'ютерами та пов'язаними з ними пристроями, роботою з різними видами інформації. Програми дають можливість взаємодіяти людині з комп'ютером та технологіями, забезпечуючи зручність та функціональність.

Замовник — це фізична або юридична особа, яка розпоряджається грошовими коштами та замовляє певні товари, роботи або послуги (цінності), або подає заявку на придбання або замовлення товарів, робіт або послуг (цінностей) у майбутньому.

У результаті виконання дипломної роботи буде розроблено програмне забезпечення (сайт) з реалізованими правами доступу для замовників та розробників програмного забезпечення.

Сайт — це набір електронних документів (файлів), які належать приватній особі або організації і знаходяться в комп'ютерній мережі, об'єднані під однією адресою (доменним ім'ям або IP-адресою).

Існує багато різновидів сайтів, які можна класифікувати за різними категоріями. Залежно від доступних сервісів, сайти можна розділити на:

- відкриті — всі сервіси сайту є доступними для всіх відвідувачів;
 - напіввідкриті — для доступу необхідно реєструватися (зазвичай, безкоштовно);
 - закриті — є доступними для вузького кола користувачів.
- Зазвичай, це службові сайти організацій (корпоративні сайти), особисті сайти приватних осіб тощо. Доступ для нових людей, зазвичай, надається через запрошення (інвайти).

За фізичним розташуванням сайти поділяються на:

- зовнішні сайти мережі Інтернет;
- локальні сайти - є доступними тільки в межах локальної мережі.

Це можуть бути як корпоративні сайти організацій, так і сайти приватних осіб в локальній мережі провайдера [2].

За схемою представлення інформації, її об'єму і категорії вирішуваних завдань:

- інтернет-портал;
- інформаційні ресурси;
- інтернет-представництва власників бізнесу;
- вебсервіс;
- комбіновані вебсервіси.

Створений сайт задовольнятиме критерії зовнішнього закритого інтернет-представництва власників бізнесу.

1.2 Огляд програмних аналогів

У зв'язку зі зміною ринкових умов і росту важливості віддаленої роботи, організація робочих проєктів та комунікації шляхом використання централізованої платформи стає надзвичайно важливою. Спеціальне програмне забезпечення для керування проєктами має велике значення,

оскільки його основна мета полягає у сприянні всім учасникам розробки в плануванні проєкту, делегуванні завдань, встановленні строків та етапів виконання, документуванні процесів, підтримці комунікації в одному централізованому місці і оцінці прогресу проєкту.

На разі представлено багато програмного забезпечення, яке забезпечує практичний спосіб відстеження проєктів через візуальну платформу. Двоє з найпопулярніших – це Trello та Jira [3]. Обидві програми належать австралійській програмній компанії Atlassian. Вони мають схожі характеристики, кожна із них має свою особливість, що робить їх найкращими для різних типів проєктів.

Хоча Trello та Jira пропонують першокласні інструменти керування завданнями, кожен має різні переваги. Найбільша різниця між ними полягає в тому, що Trello пропонує лише дошки Kanban, тоді як Jira пропонує дошки Kanban, Scrum і спринт. Інтерфейс кожного програмного забезпечення показано на рисунках 1.1 – 1.2.

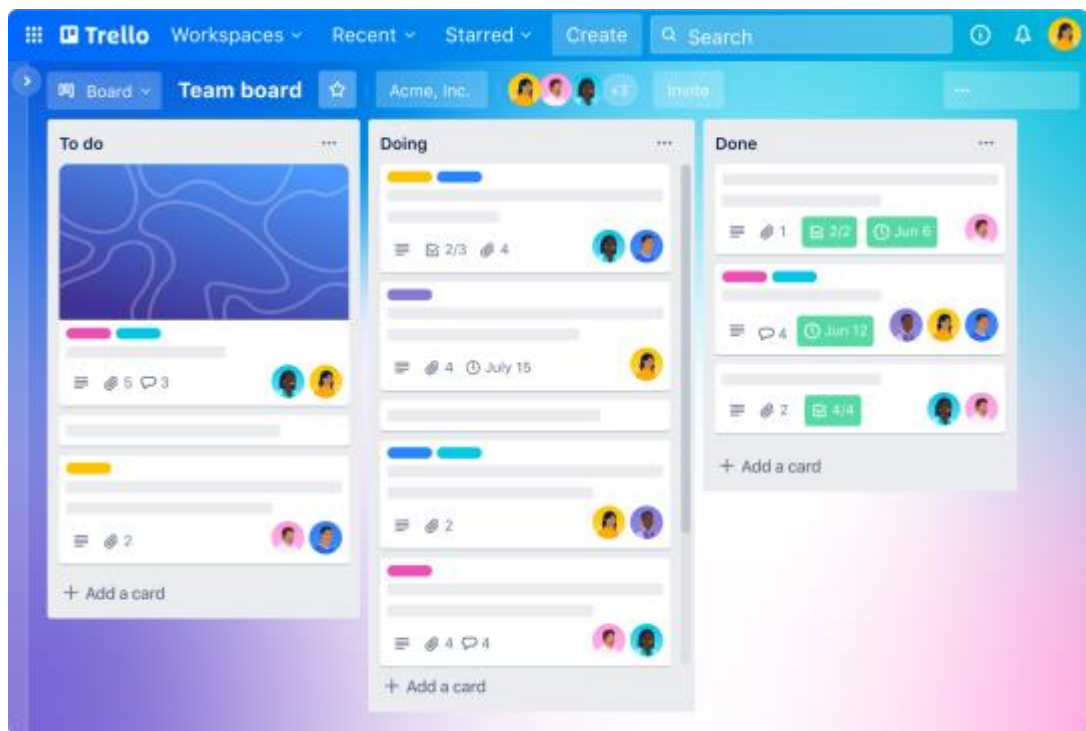


Рисунок 1.1 – Інтерфейс Trello

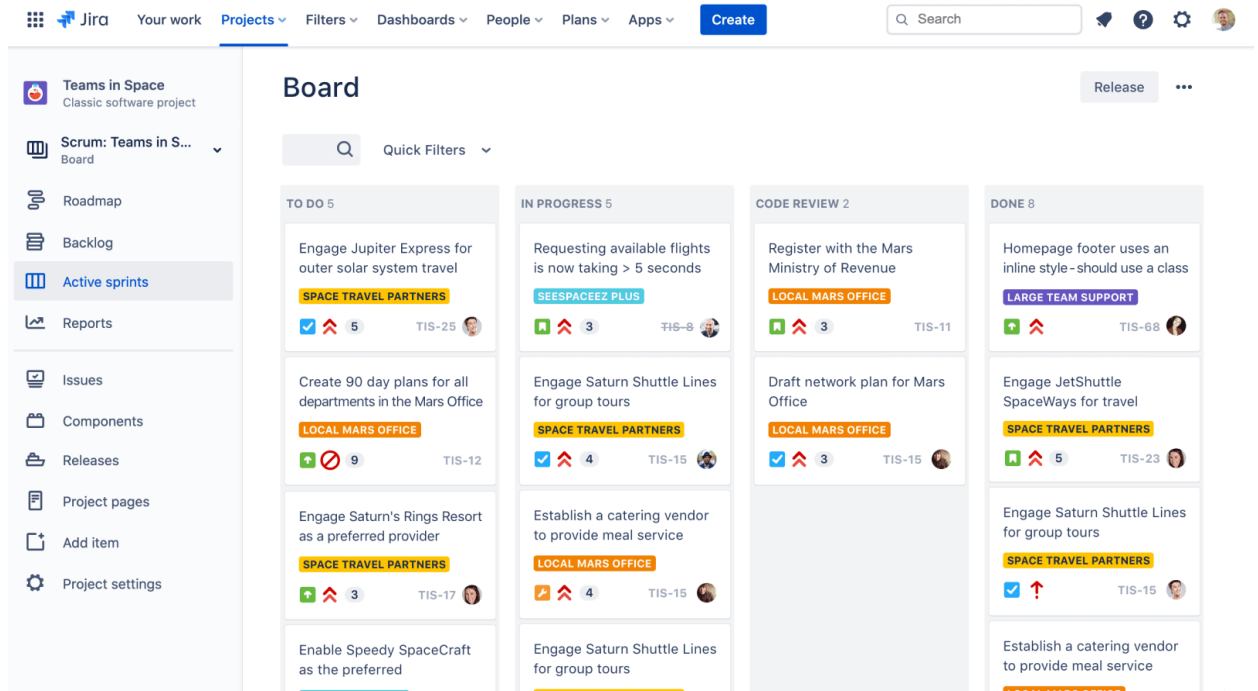


Рисунок 1.2 – Інтерфейс Jira

Дошки Scrum і Kanban використовуються для візуального відстеження завдань проєкту. Scrum зосереджується на невеликих приростах роботи, виконаних у «спринтах» або коротких часових рамках. Scrum широко використовується групами розробників програмного забезпечення. Складні завдання розбиваються на менші завдання, які потрібно виконати в спринті. Scrum дошки дотримуються дуже специфічної та жорсткої методології. В той же час, дошки Kanban розроблені, щоб допомогти візуалізувати робочий процес. Це більш гнучка система, яка легко адаптується до мінливих пріоритетів проєкту, та є більш зрозумілою для «нетехнічних» учасників проєкту.

Розглянемо також менш відомий аналог як Notion, інтерфейс якого показано на рисунку 1.3.

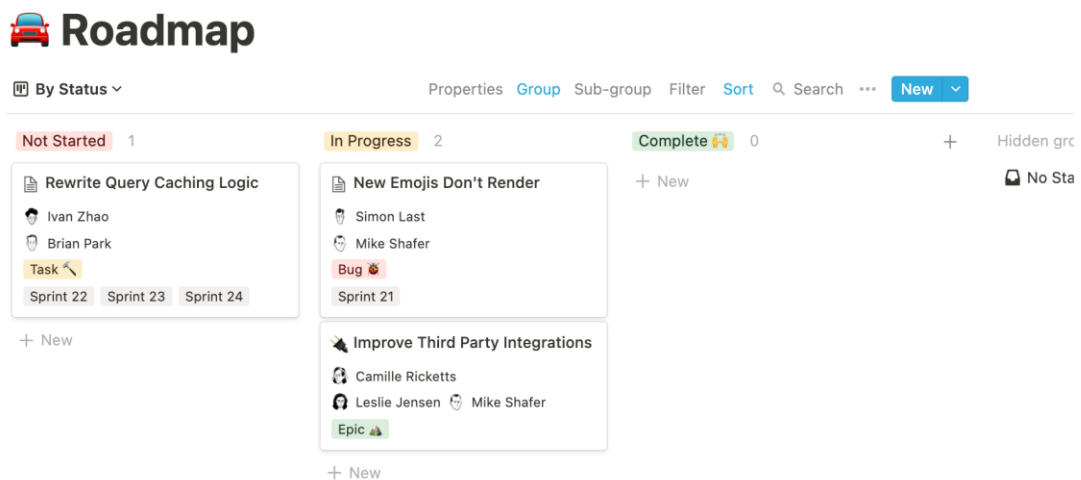


Рисунок 1.3 – Інтерфейс Notion

Notion служить ресурсним центром компанії з каталогом персоналу, сторінкою місії та цілями компанії [4]. Trello спрощує проекти у невеликі завдання, дозволяючи призначати та організовувати роботу. Там, де Trello та Jira організовують всіх членів команди, Notion поєднує команди та окремі групи, щоб переконатися, що всі працюють на одній сторінці.

Notion — це більше система керування файлами на основі шаблонів, яка включає часові шкали, діаграми Ганта, параметри бюджетування, виставлення рахунків. Trello та Jira добре підходить як інструмент для управління проектами, що відповідає робочому процесу в стилі Канбан. Користувачі перетягують завдання через конвеєр для керування проектом.

Таким чином, Trello є чудовим варіантом для компаній, яким потрібен універсальний пакет програмного забезпечення для керування проектами. Його гнучкість дозволяє застосовувати можливості в широкому діапазоні галузей, типів проектів і розмірів команд. Jira має більшу стабільність розширених функцій. Вона пропонує повний набір засобів відстеження помилок, враховує проекти в різних випусках і дозволяє користувачам

налаштовувати програмне забезпечення відповідно до потреб і цілей організації. Але через розширені параметри налаштування, що пропонуються, вона не така інтуїтивно зрозуміла у використанні, як Trello. Notion призначений для компаній, яким потрібен більш динамічний спосіб управління та розробки вмісту, тоді як Trello та Jira відображають завдання та інтегрують їх у проєктній машині.

Отже, Trello — це ідеальний інструмент для компаній, яким потрібен універсальний пакет програмного забезпечення для керування проєктами. Це гнучко та легко відповідає організаційним потребам більшості звичайних підприємств. Jira, з іншого боку, пропонує функції, які більше підходять для традиційних команд розробників програмного забезпечення, такі як Agile-менеджмент проєктів і відстеження помилок. Notion має на меті візуалізацію роботи над проєктом за допомогою наочних показників.

1.3 Висновки за розділом 1

Перший розділ направлений на аналіз предметної області, в рамках якого були визначені основні поняття і виявлена цінність розроблюваного програмного забезпечення. Також були визначені основні завдання, які воно повинно вирішувати.

Після аналізу конкурентів можна зробити висновок, що для автоматизації взаємодії між замовником і розробником програмного забезпечення необхідно поєднання всіх можливостей аналогічних рішень. Характеристики проєкту мають бути доступними як виконавцям проєкту, так і замовникам, при цьому повинні повністю задовольняти потреби обох сторін і не створювати додаткових складнощів при використанні та розумінні. Також слід враховувати потребу в конфіденційності даних, тому програмне забезпечення повинно мати чіткі обмеження доступу. Крім того, жодне з розглянутих програмних продуктів не надає можливості взаємодії між

різними учасниками процесу. Розроблюваний вебзастосунок має вирішувати поставлені завдання.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір засобів проєктування і розробки

Оскільки створення даного вебзастосунку має певну специфіку – передбачає роботу з великими за обсягом наборами даних, таких як: проєкти, завдання та користувачі, з метою накопичення та обробки цих даних, було обрано в якості основної мови розробки – JavaScript [5], яка на даний момент є дуже популярною, має доведену ефективність. Нижче наведено аргументацію даного вибору.

Для розробки серверної частини сайту використовуються різні мови програмування. Ось кілька популярних мов, які широко використовуються для цього:

- JavaScript: JavaScript є однією з найпоширеніших мов програмування, яка використовується як на клієнтській, так і на серверній стороні. За допомогою серверного середовища Node.js, JavaScript може бути використаний для розробки серверних застосунків та API;

- Python: Python є мовою програмування з простим та зрозумілим синтаксисом. Він широко використовується для розробки серверних застосунків, вебфреймворків та API. Python має велику кількість бібліотек та фреймворків, таких як Django та Flask, що полегшують розробку вебзастосунків;

- Ruby: Ruby є мовою програмування, яка часто використовується для розробки вебзастосунків та серверних API. Фреймворк Ruby on Rails є дуже популярним для швидкої та ефективної розробки вебзастосунків;

- Java: Java є мовою програмування, яка забезпечує високу міцність, стабільність та масштабованість. Вона широко використовується для розробки великих корпоративних застосунків, включаючи серверну частину вебсайтів та вебсервісів;

— PHP: PHP є мовою програмування, спеціально розробленою для розробки вебзастосунків. Вона широко використовується для розробки серверної частини сайтів, особливо на популярних платформах, таких як WordPress та Drupal.

Вибір мови програмування для серверної розробки залежить від конкретних потреб проекту, командних навичок та особистих вподобань розробників. За цієї особливості постало питання вибору між JavaScript та Python. Порівняння цих двох мов наведено у таблиці 2.1.

Таблиця 2.1 – Порівняння JavaScript та Python

Особливості	JavaScript	Python
Синтаксис	Орієнтований на об'єкти	Чистий та легко зрозумілий
Популярність	Широко використовується на клієнтській стороні та сервері за допомогою Node.js	Популярність як мови розробки вебзастосунків
Доступність	Вбудована мова у веббраузери	Встановлення окремо, але широко підтримується
Асинхронність	Підтримується з використанням промісів та асинхронних функцій (async/await)	Підтримується з використанням бібліотек, таких як asyncio
Вебфреймворки	Express.js, Koa.js, Nest.js та інші	Django, Flask, Pyramid та інші
Швидкодія	Швидкість виконання може залежати від оптимізації коду та використання розширень	Зазвичай має високу швидкодію, особливо у комбінації з популярними фреймворками

Продовження таблиці 2.1

Особливості	JavaScript	Python
Типізація	Динамічна типізація	Динамічна типізація, але також має можливість використовувати статичну типізацію через бібліотеки, такі як <code>mypy</code>
Використання	Широко використовується для розробки фронтенду, а також серверної частини з використанням <code>Node.js</code>	Зазвичай використовується для серверної розробки, а також веброботи
Екосистема	Велика кількість пакетів та бібліотек на <code>npm</code>	Велика кількість пакетів та бібліотек на <code>PyPI</code>
Стандартні бібліотеки	Широкий набір стандартних бібліотек для різних цілей	Має розширену стандартну бібліотеку з численними модулями

JavaScript необов'язково є кращим вибором ніж Python для розробки серверної частини сайту. Обидві мови мають свої переваги та використовуються широко у веброботі. Однак, ось декілька аргументів, які можуть підкреслити переваги JavaScript для розробки серверної частини:

— Єдність мови: JavaScript може бути використаний як мова ядра як на стороні клієнта, так і на стороні сервера (з допомогою `Node.js`). Використання однієї мови на всьому стеку може спростити розробку, управління та обслуговування проєкту, оскільки команда розробників може використовувати одну мову та переходити між різними компонентами проєкту з меншими зусиллями. Оскільки розробка застосування здійснюється 2 програмістами, клієнтську частину реалізовано за допомогою `React`. Цей аргумент є найбільш важливим;

— Асинхронність та події: JavaScript має потужну підтримку асинхронного програмування та обробки подій, що є важливим для серверних застосувань. Завдяки таким інструментам, як проміси та асинхронні функції (`async/await`), можна ефективно обробляти багато запитів одночасно та реагувати на події в реальному часі;

— Швидкодія: JavaScript у поєднанні з Node.js може мати високу швидкодію завдяки однопотоковій, неблокуючій I/O моделі. Це дозволяє ефективно обробляти багато одночасних запитів та забезпечувати високу продуктивність сервера;

— Велика екосистема: JavaScript має широку та активну екосистему з великою кількістю пакетів, модулів та фреймворків, доступних через пакетний менеджер `npm`. Це дає розробникам доступ до готових рішень, які можна використовувати для розширення функціональності серверної частини та прискорення розробки.

В якості платформи для серверної частини застосування було обрано Node.js.

Node.js — середовище виконання JavaScript, що дає змогу запускати код JS на комп'ютері в якості окремого застосування.

Однією з основних особливостей Node.js є використання однопотокової асинхронної моделі з циклом подій. У цій моделі завдання виконуються послідовно в одному потоці, що дозволяє ефективно використовувати ресурси. У порівнянні з багатопотоковими підходами, це спрощує програмування, оскільки завдання чергуються, коли одне чекає на виконання, інше може рухатись вперед. Хоча в однопроцесорних системах завдання все одно виконуються послідовно, використовуючи потоки, асинхронна модель дозволяє ефективно керувати виконанням завдань навіть на багатопроцесорних системах.

Механізм подій дозволяє серверу обробляти запити без блокування, що забезпечує високу масштабованість порівняно з традиційними серверами, які створюють окремий потік для кожного запиту, що призводить до

високого споживання пам'яті та ресурсів процесора. Використання однопотокової моделі дозволяє Node.js обслуговувати більше запитів одночасно, ніж традиційні сервери.

У Node.js всі функції виконуються в фоновому режимі без блокування потоку, оскільки всі бібліотеки Node.js є асинхронними. Ця асинхронна модель особливо підходить для програм, в яких багато користувачів виконують дії, що не навантажують процесор. Тому сервери, побудовані на Node.js, ефективно працюють з завданнями, які не є важкими обчислювальними процесами.

Підхід Node.js передбачає помітне підвищення продуктивності в порівнянні з іншими технологіями, значно спрощує роботу, де є велика кількість HTTP запитів до внутрішніх та зовнішніх сервісів. До того ж сервери на Node.js добре виконують роль посередників в клієнт-серверній системі [6].

У подальшому виникло питання вибору конкретного вебфреймворку у роботі з Node.js. Було 2 варіанти: Express і Коа. За сукупністю показників, що наведені у таблиці 2.2, було обрано Коа.

Таблиця 2.2 – Вибір фреймворку розробки вебзастосунку

Критерій	Коа	Express
Зручна робота з даними	Так	Так
Функції асинхронізації	Так	Ні
Просте налаштування роутингу	Так	Так
Величезне ком'юніті та підтримка фреймворка	Так	Так
Зручна робота з даними	Так	Так

2.2 Визначення функціональних вимог до програмного забезпечення

Програмне забезпечення для автоматизації роботи між замовником і розробником програмного забезпечення повинно надавати можливість роботи з цілим набором функціоналу, серед яких наступні пункти:

- створення проєктів розробника;
- виведення усіх проєктів користувача;
- переведення проєктів зі статусу «Active» у «Archive» та навпаки;
- редагування даних проєкту;
- швидке маніпулювання даними (редагування, додавання, видалення інформації);
- обчислення часу секцій розробки;
- можливість додавання файлів до проєкту;
- зміна налаштувань користувача;
- зміна налаштувань проєкту;
- робота чату для обміну інформацією між клієнтом та розробником.

У відповідності з даними функціональними можливостями визначено можливості роботи замовника, зазначеного у переліку, а також додатково відображено на рисунку 2.1:

- перегляд усіх проєктів та їх налаштувань;
- можливість завантаження необхідних файлів з проєкту;
- використання чату для обміну інформацією.



Рисунок 2.1 – Схема можливостей замовника

Можливості роботи розробника зазначено у переліку нижче, а також відображено на рисунку 2.2:

- перегляд усіх проєктів;
- редагування даних проєкту;
- обчислення часу секцій розробки;
- можливість додавання необхідних файлів до проєкту;
- зручне та швидке маніпулювання даними (редагування, додавання, видалення інформації);
- зміна власних налаштувань;
- зміна налаштувань проєкту;
- використання чату для обміну інформацією.



Рисунок 2.2 – Схема можливостей розробника

2.3 Проектування бази даних

Сьогодні існують наступні види баз даних: реляційні та нереляційні. Розглянемо їх більш детально.

NoSQL бази даних, на відміну від реляційних (SQL) сховищ, пропонують інший підхід до зберігання даних. Вони є документо-орієнтованими і нагадують папки, що містять файли. Кожна технологія NoSQL має свою власну модель даних. Ці бази даних дозволяють краще масштабуватися, не мають обмежень щодо типів даних, які можна зберігати разом, і дозволяють додавати нові типи, якщо це необхідно. Вони також пропонують швидшу обробку даних.

Проте, варто враховувати, що нереляційні бази даних не гарантують синхронізацію даних між різними вузлами, що може створювати ризик неконсистентності. Також вони не надають такої ж простоти, стійкості та сумісності, як реляційні сховища даних.

Отже, хоча NoSQL бази даних мають свої переваги у синхронізації та швидкодії, реляційні сховища є простими, стійкими, гнучкими, продуктивними, масштабованими та сумісними, але можуть стати незручними при зростанні кількості застосувань та обсягу даних.

Переваги SQL бази даних:

- атомарність, узгодженість, ізолюваність, довговічність;
- кожна таблиця містить одну або декілька категорій даних у стовпцях;
- кожен рядок містить унікальний екземпляр даних для категорій, визначених стовпцями;
- користувач може мати доступ до даних з бази даних, не знаючи структури таблиці.

Недоліки SQL бази даних:

- необхідність у масштабуванні на потужних серверах;
- необхідність ускладнювати систему, що робить обробку більш важкою;
- дорогі системи зберігання та запатентованих серверів.

Переваги NoSQL бази даних:

- зберігання погано структурованих великих обсягів даних;
- швидка обробка даних;
- значно дешевші в обслуговуванні, ніж реляційні;
- краща масштабованість [7].

Недоліки NoSQL бази даних:

- узгодженість даних, ризики синхронізації вузлів;
- відсутність стандартизації;
- погано розповсюджуються на декілька вузлів.

В якості бази даних для застосування було обрано NoSQL базу даних MongoDB [8], бо є необхідність зберігати і структурувати інформацію, що має відмінний тип для різних об'єктів.

Переваги MongoDB:

- відсутність схеми: дана БД заснована на колекціях різних документів. Кількість полів, зміст і розмір цих документів може відрізнятися;
- зрозуміла структура кожного об'єкта;
- легко масштабується;
- зберігання використовуваних в даний момент даних здійснюється за допомогою внутрішньої пам'яті, що дозволяє отримувати більш швидкий доступ;
- дані зберігаються у вигляді JSON документів;
- MongoDB підтримує динамічні запити документів (document-based query);
- відсутність складних JOIN запитів.

Недоліки MongoDB:

- немалий розмір даних;
- відсутність звичного JOIN запиту;
- недосконала система Map/Reduce [9].

Для розробки даного вебзастосунку було обрано документо-орієнтовану модель даних. Для реалізації цієї моделі використовувалася MongoDB.

Основною відмінністю документо-орієнтованих баз даних від традиційних реляційних є їх структура зберігання даних. У документо-орієнтованих базах даних, дані зберігаються у вигляді колекцій, які можуть містити різноманітні об'єкти з різною структурою і набором властивостей. Натомість, у реляційних базах даних, дані організовані у вигляді однотипних і жорстко структурованих рядків.

У документо-орієнтованих базах даних, колекція є іменованим набором об'єктів, при цьому кожен об'єкт належить лише одній колекції. Кожен об'єкт включає унікальний ідентифікатор `_id` і може містити різні властивості. Властивість представляє собою пару з назви, типу та значення.

Документо-орієнтовані бази даних мають різноманітні типи, такі як: рядок, ціле число, число з плаваючою точкою, масив, об'єкт, бінарний рядок, байт, символ, дата, булеве значення і null.

Завдяки такій структурі зберігання, документо-орієнтовані бази даних зазвичай є швидшими у порівнянні з традиційними реляційними базами даних, оскільки дані можуть бути збережені у вигляді цілих документів, які можуть бути одразу отримані та змінені без необхідності виконання складних операцій з'єднання таблиць.

Підтримуються операції вибірки та агрегації (Aggregation Pipeline, Map-Reduce, count, distinct), вставки, зміни та видалення. Об'єкти не мають зв'язків, але можуть містити інші об'єкти та окремі властивості інших об'єктів у своїх властивостях. Наявна підтримка унікальних та композитних індексів, які можна накладати на властивості вкладених об'єктів.

У багатьох документо-орієнтованих базах даних, таких як MongoDB, використовується 12-байтний унікальний ідентифікатор, який називається ObjectId. Цей ідентифікатор зазвичай складається з різних компонентів, таких як час створення, ідентифікатор комп'ютера та унікальний лічильник. В результаті отримується значення, яке гарантує практичну унікальність в межах бази даних.

Оскільки ObjectId має достатньо широкий діапазон значень і забезпечує розподіленість ідентифікаторів, це дозволяє незалежно вставляти дані на різних машинах без конфліктів або переповнення числа. Кожен комп'ютер може генерувати свої унікальні ObjectId, і ці значення будуть унікальними в межах бази даних.

Таким чином, документо-орієнтовані бази даних, які використовують ObjectId, забезпечують ефективну синхронізацію реплік і дозволяють незалежно вставляти дані на різних машинах без необхідності вирішувати проблему переповнення цілого числа.

В процесі проєктування даного програмного продукту для побудови об'єктно-орієнтованої моделі було виділено наступні об'єктні множини:

«user», «client», «estimator», «project», «section», «task», «settings», «attachment», «message», «additional», «library», «librarySection», «libraryItem», «token», «emailToken», «clientToken». Аналіз визначених об'єктів і атрибутів дозволяє виділити сутності бази даних і побудувати її логічну схему.

Спроектowana об'єктно-орієнтована модель зображена на рисунку 2.3.

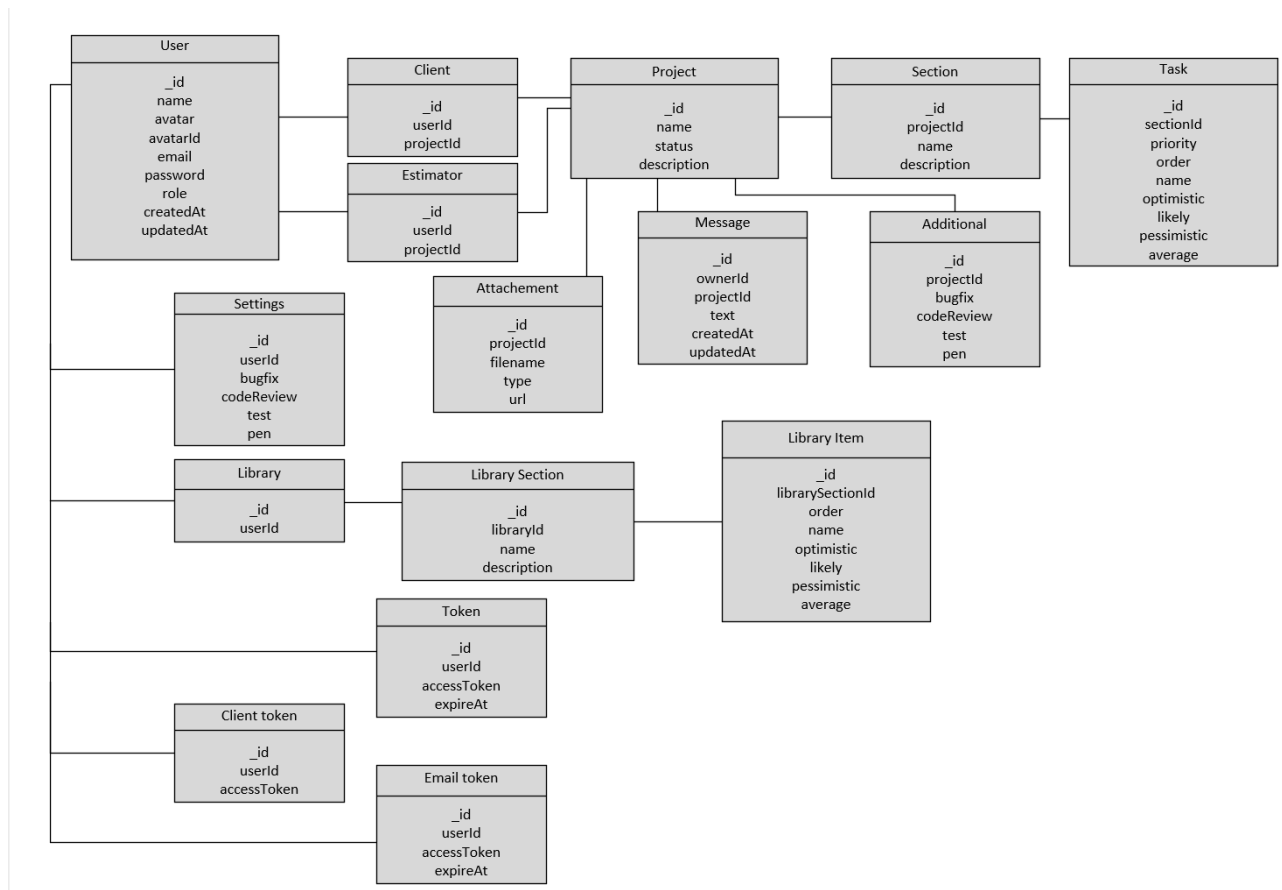


Рисунок 2.3 – Спроектowana об'єктно-орієнтована модель

2.4 Визначення структури програми

Структурно серверна частина складається з наступних модулів:

- actions – описані основні події;
- components – описані основні компоненти застосування;
- config – містяться компоненти конфігурації ПЗ;
- middlewares – описані обробники даних;
- model – міститься опис моделей БД;

— validator – описані функції валідації даних перед і після роботи з БД.

Роутери з `index.js` отримують HTTP-запити від клієнту і пересилають їх на обробники `middlewares`, які перенаправляють дані у функції з `actions` для роботи з базою даних, використовуючи моделі, описані у каталозі `model`, визначені бібліотекою `Mongoose` [10]. Інформація про користувачів, проєкти, налаштування знаходиться в базі даних `MongoDB`. При необхідності робиться перевірка вхідних даних перед роботою з БД за допомогою `validator`. При кожній дії користувача перевіряється токен доступу за допомогою `jwt`-стратегії. Обробники `middlewares` отримують дані з `actions`, формуючи HTTP-відповіді, передаючи їх роутеру, який у свою чергу відправляє дану відповідь до клієнтської частини.

Структурна схема сервера зображена на рисунку 2.4.

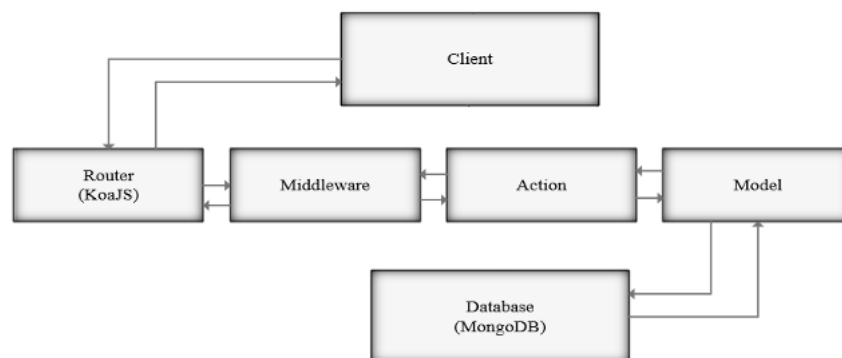


Рисунок 2.4 – Структурна схема сервера

Взаємодія між клієнтом і сервером відбувається за допомогою REST (Representational State Transfer) архітектурного стилю. REST є підходом до розробки розподілених систем, який заснований на принципах використання простих і стандартних протоколів HTTP [11].

Кожна одиниця інформації, така як ресурс або об'єкт, в REST має своє унікальне посилання, відоме як URL (Uniform Resource Locator). Це посилання ідентифікує ресурс і використовується клієнтом для отримання доступу до нього.

Дії над даними визначаються за допомогою методів HTTP запитів. Основними методами, що використовуються в REST, є:

1. GET: використовується для отримання ресурсу з сервера. Наприклад, клієнт може зробити GET-запит до певного URL для отримання даних про цей ресурс.
2. POST: використовується для створення нового ресурсу на сервері. Клієнт може виконати POST-запит до відповідного URL, надіславши дані, які сервер повинен зберегти як новий ресурс.
3. PUT: використовується для оновлення існуючого ресурсу на сервері. Клієнт може виконати PUT-запит до відповідного URL, надіславши нові дані, які сервер повинен використовувати для оновлення ресурсу.
4. DELETE: використовується для видалення ресурсу з сервера. Клієнт може виконати DELETE-запит до URL, що ідентифікує ресурс, який треба видалити.

Ці методи HTTP запитів визначають дії, які клієнт може виконати над ресурсами сервера, дозволяючи здійснювати стандартизовану та просту взаємодію між клієнтом і сервером у вебзастосуваннях, побудованих згідно з REST-архітектурою.

2.5 Висновки за розділом 2

У другому розділі пояснювальної записки проведено аналіз фреймворків Koa та Express для створення серверної частини. Після ретельного вивчення, обрано фреймворк Koa як найбільш популярний та ефективний інструмент для розробки вебзастосунків мовою JavaScript для платформи Node.js. Крім того, виявлено основні сценарії взаємодії між замовником і розробником у контексті вебзастосунку. Також була побудована модель бази даних та визначена структура програми, які стануть основою для подальшої розробки.

З РОЗРОБКА ПРОГРАМИ

3.1 Алгоритм функціонування програми

Програма надає різну функціональність користувачам, залежно від їх ролі. Нижче описано логіку роботи для кожної з ролей - розробника та замовника, щоб визначити алгоритм функціонування програми.

Розробник має доступ до основних функцій програми. Вони включають створення, редагування та видалення проєктів. Розробник може визначати час, витрачений на певні секції розробки, додавати файли до проєкту, змінювати статус проєкту та налаштування, а також використовувати чат для обміну інформацією з іншими користувачами. Також розробник може створювати власний профіль і змінювати різні налаштування, такі як пароль, електронна пошта, аватар та інше.

Замовник має обмежений доступ до програми. Він може переглядати інформацію про проєкти, завантажувати файли та використовувати чат для спілкування з розробником. Замовник може лише авторизуватися в системі за допомогою посилання, яке надсилається на його електронну пошту.

Ця функціональність програми забезпечує розробникам та замовникам зручні інструменти для співпраці та взаємодії в рамках вебзастосунку.

3.2 Опис реалізованих складових вебзастосунку

Вебзастосунок для автоматизації роботи між замовником і розробником має складну структуру складових серверної частини. Розглянемо більш детально з яких документів (об'єктів) воно складається.

Документ «user» (_id, username, email, password, avatar, avatar_id, role, createdAt, updatedAt) зберігає інформацію про облікові записи користувачів. Опис документа «user» представлений в таблиці 3.1.

Таблиця 3.1 – Опис документа «user»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
username	String	Унікальне ім'я користувача
email	String	Дані про поштову адресу користувача
password	String	Пароль для входу в акаунт
avatar	String	Дані про аватар користувача
avatar_id	String	Унікальний ідентифікатор аватару
role	String	Роль користувача
createdAt	Date	Дата створення облікового запису
updatedAt	Date	Дата оновлення облікового запису

Документ «client» (_id, userId, projectId) зберігає інформацію про клієнтів. Опис документа «client» представлений в таблиці 3.2.

Таблиця 3.2 – Опис документа «client»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
userId	Schema.Types.ObjectId	Ідентифікатор користувача з документа user
projectId	Schema.Types.ObjectId	Ідентифікатор проєкту з документа project

Документ «estimator» (_id, userId, projectId) зберігає інформацію про розробників. Опис документа «estimator» представлений в таблиці 3.3.

Таблиця 3.3 – Опис документа «estimator»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
userId	Schema.Types.ObjectId	Ідентифікатор користувача з документа user
projectId	Schema.Types.ObjectId	Ідентифікатор проєкту з документа project

Документ «project» (_id, name, status, descriptions, date, updatedAt) зберігає інформацію про проєкти. Опис документа «project» представлений в таблиці 3.4.

Таблиця 3.4 – Опис документа «project»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
name	String	Назва проєкту
status	String	Статус проєкту
descriptions	String	Опис проєкту
date	Date	Дата створення проєкту
updatedAt	Date	Дата оновлення проєкту

Документ «section» (_id, projectId, name, description) зберігає інформацію про секції розробки проєкту. Опис документа «section» представлений в таблиці 3.5.

Таблиця 3.5 – Опис документа «section»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
projectId	Schema.Types.ObjectId	Ідентифікатор проєкту з документа project
name	String	Назва секції
description	String	Опис секції

Документ «task» (_id, sectionId, priority, order, name, optimistic, likely, pessimistic, average) зберігає інформацію про пункти секції. Опис документа «task» представлений в таблиці 3.6.

Таблиця 3.6 – Опис документа «task»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
sectionId	String	Ідентифікатор секції з документа section
priority	String	Дані про пріоритетність завдання
order	Number	Дані про положення завдання у секції
name	String	Назва завдання
optimistic	Number	Оптимістичне значення
likely	Number	Середнє значення
pessimistic	Number	Песимістичне значення
average	Number	Середнє арифметичне оптимістичного, середнього та песимістичного значень

Документ «settings» (_id, userId, bugfix, codereview, test, pen, updatedAt) зберігає інформацію про базові налаштування розробки проєктів,

що належать певному розробнику. Опис документа «settings» представлений в таблиці 3.7.

Таблиця 3.7 – Опис документа «settings»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
userId	Schema.Types.ObjectId	Ідентифікатор користувача з документа user
bugfix	Number	Значення відсотку для розрахунку часу на редагування багів
codereview	Number	Значення відсотку для розрахунку часу на перевірку коду
test	Number	Значення відсотку для розрахунку часу на тестування проєкту
pen	Number	Значення відсотку для розрахунку часу для менеджменту
updatedAt	Date	Дата оновлення налаштувань

Документ «settings» (_id, userId, bugfix, codereview, test, pen, updatedAt) зберігає інформацію про базові налаштування розробки проєктів, що належать певному розробнику. Опис документа «settings» представлений в таблиці 3.8.

Таблиця 3.8 – Опис документа «settings»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
userId	Schema.Types.ObjectId	Ідентифікатор користувача з документа user

Продовження таблиці 3.8

Назва ключа	Тип значення	Опис
bugfix	Number	Значення відсотку для розрахунку часу на редагування багів
codereview	Number	Значення відсотку для розрахунку часу на перевірку коду
test	Number	Значення відсотку для розрахунку часу на тестування проєкту
pen	Number	Значення відсотку для розрахунку часу для менеджменту
updatedAt	Date	Дата оновлення налаштувань

Документ «attachment» (_id, projectId, fileName, type, url, attachment_id) зберігає інформацію про закріплені до проєкту файли. Опис документа «attachment» представлений в таблиці 3.9.

Таблиця 3.9 – Опис документа «attachment»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
projectId	Schema.Types.ObjectId	Ідентифікатор проєкту з документа project
fileName	String	Назва закріпленого файлу
type	String	Тип закріпленого файлу
url	String	Посилання на файл
attachment_id	String	Забезпечує унікальність файлу на сховищі

Документ «message» (_id, projectId, ownerId, text, createdAt, updatedAt) зберігає інформацію про повідомлення. Опис документа «message» представлений в таблиці 3.10.

Таблиця 3.10 – Опис документа «message»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
projectId	Schema.Types.ObjectId	Ідентифікатор проєкту з документа project
ownerId	Schema.Types.ObjectId	Ідентифікатор користувача з документа user
text	String	Текст повідомлення
createdAt	Date	Дата створення повідомлення
updatedAt	Date	Дата редагування повідомлення

Документ «additional» (_id, projectId, bugfix, codereview, test, pen) зберігає інформацію про налаштування проєкту. Опис документа «additional» представлений в таблиці 3.11.

Таблиця 3.11 – Опис документа «additional»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
projectId	Schema.Types.ObjectId	Ідентифікатор проєкту з документа project
bugfix	Number	Значення відсотку для розрахунку часу на редагування багів
codereview	Number	Значення відсотку для розрахунку часу на перевірку коду
test	Number	Значення відсотку для розрахунку часу на тестування проєкту
pen	Number	Значення відсотку для розрахунку часу для менеджменту

Документ «library» (_id, userId) зберігає інформацію про бібліотеки секцій користувача. Опис документа «library» представлений в таблиці 3.12.

Таблиця 3.12 – Опис документа «library»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
userId	Schema.Types.ObjectId	Ідентифікатор користувача з документа user

Документ «librarySection» (_id, libraryId, name, description) зберігає інформацію про секції бібліотек користувача. Опис документа «librarySection» представлений в таблиці 3.13.

Таблиця 3.13 – Опис документа « librarySection»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
libraryId	Schema.Types.ObjectId	Ідентифікатор бібліотеки з документа library
name	String	Назва секції бібліотеки
description	String	Опис секції бібліотеки

Документ «libraryItem» (_id, librarySectionId, order, name, optimistic, likely, pessimistic, average) зберігає інформацію про пункти секцій бібліотек користувача. Опис документа «libraryItem» представлений в таблиці 3.14.

Таблиця 3.14 – Опис документа «libraryItem»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа

Продовження таблиці 3.14

Назва ключа	Тип значення	Опис
librarySectionId	String	Ідентифікатор секції з документа librarySection
order	Number	Дані про положення пункту у секції
name	String	Назва пункту
optimistic	Number	Оптимістичне значення
likely	Number	Середнє значення
pessimistic	Number	Песимістичне значення
average	Number	Середнє арифметичне оптимістичного, середнього та песимістичного значень

Документ «token» (_id, userId, accessToken, expireAt) зберігає інформацію про токени доступу користувача. Опис документа «token» представлений в таблиці 3.15.

Таблиця 3.15 – Опис документа «token»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
userId	Schema.Types.ObjectId	Ідентифікатор користувача з документа user
accessToken	String	Дані про токен доступу
expireAt	Date	Термін придатності токenu

Документ «emailToken» (_id, userId, accessToken, expireAt) зберігає інформацію про токени доступу для зміни пароля користувача. Опис документа «emailToken» представлений в таблиці 3.16.

Таблиця 3.16 – Опис документа «emailToken»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
userId	Schema.Types.ObjectId	Ідентифікатор користувача з документа user
accessToken	String	Дані про токен доступу
expireAt	Date	Термін придатності токена

Документ «clientToken» (_id, userId, accessToken) зберігає інформацію про токени доступу для клієнта. Опис документа «clientToken» представлений в таблиці 3.17.

Таблиця 3.17 – Опис документа «clientToken»

Назва ключа	Тип значення	Опис
_id	String	Забезпечує унікальність документа
userId	Schema.Types.ObjectId	Ідентифікатор користувача з документа user
accessToken	String	Дані про токен доступу

3.3 Висновки за розділом 3

Для розробки вебзастосунку було обрано документо-орієнтовану модель даних, реалізовану через MongoDB, яка надає більше гнучкості та швидкості в порівнянні з реляційними базами даних.

Процес проєктування програмного продукту включав виділення та аналіз об'єктних множин, що дозволило побудувати структурну схему бази даних.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Опис програми

Якісна взаємодія між замовником і розробником програмного забезпечення є дуже важливою, оскільки сприяє покращенню ефективності та якості розроблюваного продукту. Для спрощення цього процесу було розроблено програмне забезпечення, що є метою даної дипломної роботи.

Одним з ключових аспектів цього програмного забезпечення є автоматичне відстеження стану проєктів, облік витрат часу та вирішення конфліктних ситуацій. Це допомагає уникнути непорозумінь і сприяє підвищенню ефективності та якості виконання завдань.

Існують подібні програмні аналоги, але вони не пропонують зручних засобів роботи та зворотного зв'язку у вигляді чату. Чат є важливим елементом, оскільки забезпечує комфортну і швидку взаємодію між розробником і замовником. Саме реалізація цієї функціональності є головною метою розроблюваного застосунку.

Нижче наведені найбільш значущі алгоритми роботи серверної частини програмного продукту, які є об'єктом дослідження даної дипломної роботи.

Один з обов'язкових алгоритмів взаємодії між клієнтською і серверною частинами полягає у перевірці валідності токена, включаючи його термін дії. Ця перевірка здійснюється перед кожним запитом до сервера і забезпечує безпеку. Для цього використовується JWT (JSON Web Token) технологія.

JWT (JSON Web Token) - широко використовувана стратегія аутентифікації та перевірки дійсності в сучасних вебзастосунках. Вона забезпечує безпечний та ефективний спосіб передачі та перевірки облікових даних користувача. При вході користувача на сервері генерується JWT, який містить інформацію про користувача та підписується за допомогою секретного ключа. Цей токен потім надсилається назад до клієнта, де він

зберігається, зазвичай в локальному сховищі або в куках. При подальших запитах клієнт включає JWT в заголовок Authorization, що дозволяє серверу перевірити токен та надати доступ до захищених ресурсів. Ця стратегія усуває потребу в управлінні сесіями на серверному боці, що робить її масштабованою та підходящою для розподілених архітектур.

Здійснюється це за допомогою роуту «/checkExpireToken». Після чого з використанням функції passport.authenticate [12] перевіряється його наявність у базі даних та час його придатності. Код представлено на рисунку 4.1.

```
import passport from 'passport';
import jwt from 'jsonwebtoken';
import User from '../model/userModel';
import Token from '../model/tokenModel';
import * as _ from 'lodash';
import publicFields from '../components/db/commonFields/user';

import { Strategy } from 'passport-jwt';
import { ExtractJwt } from 'passport-jwt';

const JwtStrategy = Strategy;

export default function () {
  const opts = {}
  opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
  opts.secretOrKey = process.env.SECRET;

  passport.use(new JwtStrategy(opts, async (jwt_payload, done) => {
    const tokenRec = await Token.findOne({
      accessToken: jwt.sign(jwt_payload, process.env.SECRET),
      expireAt: {
        $gt: new Date(),
      }
    });

    if (!tokenRec) {
      await Token.deleteMany({
        expireAt: {
          $lt: new Date(),
        }
      });
      return done(null, null);
    }

    User.findOne({ _id: jwt_payload._id }, (err, user) => {
      if (err) {
        return done(err, null);
      }
      if (user) {
        return done(null, _.pick(user, publicFields));
      }
      else {
        return done(err, null);
      }
    });
  }));

  return passport.initialize()
}
```

Рисунок 4.1 – Код налаштування і роботи JWT-токена

У позитивному випадку сервер надсилає дані користувача та його налаштування. Якщо токен не знайдено у БД чи він прострочений, то надсилається повідомлення про помилку.

Для створення проєкту сервер очікує звернення із даними `clientEmail`, `name`, `projectName`, `description`, `token`, `settings` за роутом «`/createproject`». Після чого перевіряється наявність даного клієнта за поштою, у разі його відсутності створюється відповідний об'єкт у БД. Якщо є, то далі використовується його ідентифікатор користувача.

Після того як всі дані отримані і перевірені у базі даних створюється об'єкт проєкту для користувача і розробника, після чого клієнту відправляється посилання на проєкт на пошту за допомогою сервісу Mailgun.

Mailgun - це сервіс електронної пошти, який надає інструменти для відправлення, отримання та керування електронними листами через програмний інтерфейс (API). Він дозволяє легко і надійно інтегрувати функціональність електронної пошти в додатки та сайти.

За допомогою Mailgun можна відправляти електронні листи з додатку або сайту через їхні сервери. Забезпечує надійну доставку листів до поштових скриньок отримувачів і високу швидкість доставки.

Mailgun також надає розширені можливості керування електронною поштою, такі як відстеження доставки листів, автоматичне розпізнавання спаму та виявлення проблем доставки. Є можливість отримувати звіти про доставку листів, статистику електронної пошти, а також налаштовувати правила пересилання та фільтрацію листів [13].

Крім того, Mailgun надає можливості для обробки вхідних листів. Можна налаштувати поштові скриньки для отримання і обробки листів через API. Це дає можливість автоматично обробляти листи, витягувати дані з них, виконувати певні дії на основі вмісту листів та багато іншого.

Загалом, Mailgun є потужним сервісом електронної пошти, який дозволяє вам легко і надійно керувати відправкою та отриманням листів у

своїх додатках та вебсайтах. Він забезпечує швидку доставку, аналітику, керування та розширені можливості обробки електронної пошти.

У разі успішного виконання всіх умов розроблюваний сервер повертає дані проєкту, у протилежному випадку – повідомлення з помилкою.

Код роботи створення проєкту показано на рисунку 4.2.

```
import jwt from 'jsonwebtoken';
import generator from 'generate-password';

import projectModel from '../model/projectModel';
import estimatorModel from '../model/estimatorModel';
import Token from '../model/tokenModel';
import User from '../model/userModel';
import clientModel from '../model/clientModel';
import additionalModel from '../model/additionalModel';

import { sendProjectEmail } from '../actions/emailUtilsForProjects';
import { setClientToken } from '../actions/actionClientToken';

const createProject = async (ctx) => {

  const { settings } = ctx.request.body;

  const { clientEmail, name, projectName, description, token } = ctx.request.body;

  const userToken = await Token.findOne({ accessToken: token });
  const client = await User.findOne({ email: clientEmail });

  if (!client) {

    let userPassword = await generator.generate({
      length: 8,
      numbers: true,
      uppercase: false,
    });

    const findUsername = await User.findOne({ username: name });
    if (!findUsername) {
      await User.create({ username: name, email: clientEmail, password: userPassword });
    } else {
      throw ({
        status: 409,
        message: 'User with this username is already exists'
      })
    }
  }

  } else if (client.role == 'Estimator') {
    throw ({
      status: 412,
      message: 'Client cannot be the Estimator',
    })
  }

  const projectsClient = await User.findOne({ email: clientEmail });
  if (projectsClient) {

    const project = await projectModel.create({ name: projectName, descriptions: description });

    await clientModel.create({ userId: projectsClient._id, projectId: project._id });
    await estimatorModel.create({ userId: userToken.userId, projectId: project._id });

    await additionalModel.create({
      projectId: project._id,
      bugfix: settings.bugfix,
      codereview: settings.codereview,
      test: settings.test,
      pen: settings.pen,
    });

    const token = await jwt.sign({ projectsClient }, process.env.SECRET_CLIENT);
    setClientToken(projectsClient._id, token);
    sendProjectEmail(projectsClient.email, project._id, token);

    ctx.body = project;
  }
}

export default createProject;
```

Рисунок 4.2 - Код роботи створення проєкту

Більшість можливостей зміни певних налаштувань мають схожий алгоритм. Проте один – завантаження та зміна аватару користувача, – має особливості, на які варто звернути окрему увагу, тому що були використанні особливі сторонні API та модулі: Busboy та Cloudinary.

Busboy є модулем Node.js, який використовується для обробки та розбору даних, що надходять через HTTP-запити, зокрема завантаження файлів. Він забезпечує зручний і потужний спосіб отримання та обробки вхідних файлів у вебзастосунках [14].

Busboy дозволяє розбирати вхідні потоки даних та виділяти файли, що надсилаються через multipart/form-data запити, які часто використовуються для завантаження файлів на сервер. Він надає зручні інтерфейси для обробки цих файлів, отримання їхніх властивостей, таких як розмір та тип, а також збереження їх на сервері.

Завдяки Busboy можна ефективно обробляти завантаження файлів в Node.js-застосуваннях, отримувати доступ до їхнього вмісту та метаданих, і виконувати необхідні операції з цими файлами, такі як збереження на диск або обробка в пам'яті.

Cloudinary - це хмарна платформа для керування медіа-ресурсами (зображеннями, відео, аудіо) в Інтернеті. Вона надає набір послуг для завантаження, збереження, оптимізації, перетворення та доставки медіа-контенту з високою швидкістю та надійністю [15].

З Cloudinary можна завантажувати медіа-файли на їх сервери і зберігати їх у хмарному сховищі. Платформа автоматично керує масштабуванням, оптимізацією та доставкою цих ресурсів, щоб забезпечити оптимальну продуктивність та швидкість завантаження для сайту або додатку.

Основні можливості Cloudinary включають оптимізацію зображень для зменшення їхнього розміру без втрати якості, автоматичне збереження копій зображень з різними розмірами та форматами для різних пристроїв і екранів (responsive images), маніпуляцію зображеннями (обрізка, зміна

розміру, різке, поворот тощо) за допомогою простого API, а також доставку зображень з високою швидкістю через CDN (Content Delivery Network).

Крім зображень, Cloudinary також підтримує оптимізацію та керування іншими типами медіа, такими як відео та аудіо. Можна здійснювати маніпуляції з відео, такі як обрізка, зміна розміру, транскодування різних форматів та якості. Cloudinary також надає засоби для потокової доставки відео через HTTP Live Streaming (HLS) та Dynamic Adaptive Streaming over HTTP (DASH).

Загалом, Cloudinary дозволяє легко та ефективно керувати медіа-ресурсами у вебпроекті, забезпечуючи швидку доставку, оптимізацію та гнучкість обробки зображень, відео та аудіо.

Сервер очікує звернення за роутом «/setavatar» із вхідними даними токєну та завантаженням файлом за допомогою FormData. З використанням бібліотеки Busboy, файл зберігається тимчасово локально, а потім завантажується у сервіс для збереження медіа-даних Cloudinary. У разі успішного завантаження у поле аватару зберігається посилання з Cloudinary.

У разі успішного виконання всіх умов сервер повертає дані користувача (публічно доступні), у протилежному випадку – повідомлення з помилкою.

Код роботи встановлення аватару показано на рисунках 4.3 – 4.4.

```
const token = ctx.request.headers.authorization;
const user = await findUser(token);
let myImage;

let busboy = new Busboy({ headers: ctx.request.headers });
busboy.on('file', function (fieldname, file, filename, encoding, mimetype) {
  file.on('data', function (data) {
  });
  file.on('end', function () {
    myImage = {
      fieldname,
      filename,
      encoding,
      mimetype,
    };
  });
  let saveTo = path.join(os.tmpdir(), path.basename(user._id.toString()));
  file.pipe(fs.createWriteStream(saveTo));
});
busboy.on('field', function (fieldname, val, fieldnameTruncated, valTruncated, encoding, mimetype) {
});
ctx.req.pipe(busboy);
```

Рисунок 4.3 – Код збереження файлу з використанням Busboy

```

const User = await new Promise( (resolve) => {

  busboy.on('finish', async function () {
    const { filename } = myImage;

    if (!user) {
      throw({
        status: 500,
        message: 'User not found',
      })
    }

    if(user.avatar_id) {
      const destroy = util.promisify(cloudinary.uploader.destroy);
      const res = await destroy(user.avatar_id)
      .catch((error) => {
        throw(error)
      });
    }

    const upload = util.promisify(cloudinary.uploader.upload);
    const image = await upload('/tmp/' + user._id, { tags: filename })
    if (image) {
      fs.unlinkSync('/tmp/' + user._id);

      const updatedAvatar = await updateUserAvatar(user._id, image.public_id, image.secure_url);
      if (!updatedAvatar) {
        throw({
          status: 500,
          message: 'Avatar is not updated',
        })
      }

      const getUser = await findUser(token);
      if (!getUser) {
        throw({
          status: 404,
          message: 'User not found'
        })
      }

      const result = _.pick(getUser, publicFields);
      resolve(result);
    } else {
      throw({
        status: 500,
        message: 'Image is not uploaded to Cloudinary'
      })
    }
  });
});

```

Рисунок 4.4 – Код встановлення аватару

Важливою функціональною можливістю розроблюваного застосування є чат між замовником і розробником програмного забезпечення. Реалізовано його у серверній частині за допомогою Socket.io.

Socket.io - це бібліотека JavaScript, яка забезпечує двосторонню комунікацію у режимі реального часу між вебклієнтом і сервером. Вона надає зручний та потужний набір інструментів для розробки застосунків, які вимагають миттєвої передачі даних. Socket.io дозволяє встановлювати

постійне з'єднання між клієнтом і сервером, що дозволяє обмінюватися повідомленнями та подіями в режимі реального часу. Ця бібліотека є популярним інструментом для створення чатів, онлайн-ігор, колаборативних редакторів та інших, які вимагають миттєвої взаємодії між користувачами [16].

У даному коді було створено обробники подій для підключення клієнта до сервера Socket.io та обміну повідомленнями між клієнтами у визначених кімнатах. Розглянемо більш детально нижче.

Підключення та створення кімнати здійснюється автоматично на стороні сервера під час його запуску.

Обробник події `socket.on('room', (data) => { ... })` виконується, коли клієнт відправляє дані про кімнату, до якої він хоче приєднатися. Коли клієнт відправляє повідомлення у поточну кімнату виконується `socket.on('message', (data) => { ... })`. Код відправляє повідомлення до всіх клієнтів, які знаходяться у тій самій кімнаті (яка визначена ідентифікатором проєкту), за допомогою `socket.broadcast.to(room).emit('messages', 'New message')`. Також в цей момент зберігається повідомлення у базі даних з шифруванням `Bcrypt` перед збереженням.

`Bcrypt` є адаптацією алгоритму хешування, призначеного для захисту даних.

`Bcrypt` використовує солі та множинне хешування для ускладнення процесу зламу. Сіль — це додатковий випадковий рядок, яка додається до пароля перед хешуванням. Це унеможлиблює використання розширених таблиць радужних хешів та злам методом підбору [17].

У багатьох популярних мовах програмування, таких як JavaScript, Python, PHP, Java, наявні бібліотеки, які реалізують функціональність `bcrypt` і дозволяють зручно використовувати його для хешування та перевірки паролів у програмних проєктах.

Для видалення повідомлення використовуються конструкції `socket.on('deleteMessage', (data) => { ... })`. Для оновлення змісту повідомлення - `socket.on('updateMessage', (data) => { ... })`.

Фрагмент коду роботи з Socket.io показано на рисунку 4.5.

```
io.on('connection', (socket) => {  
  
  socket.on('disconnect', () => {});  
  
  let room = '';  
  
  socket.on('room', (data) => {  
    room = data;  
    socket.join(room);  
  });  
  
  socket.on('message', (data) => {  
    socket.broadcast.to(room).emit('messages', 'New message');  
    saveMessage(data);  
  });  
  
  socket.on('deleteMessage', (data) => {  
    socket.to(room).emit('deletedMessage', 'Deleted message');  
    deleteMessageById(data);  
  });  
  
  socket.on('updateMessage', (data) => {  
    socket.to(room).emit('updatedMessage', 'Updated message');  
    updateMessageText(data);  
  });  
  
});
```

Рисунок 4.5 – Фрагмент коду роботи з Socket.io

4.2 Експлуатація програми

Створена система має розподілену архітектуру, тобто складається з клієнта та сервера, між якими йде обмін інформацією. Клієнт має зручний, інтерфейс, мінімалістичний дизайн та інтуїтивно зрозумілу навігацію. Сервер, у свою чергу, забезпечує функціональність клієнта.

Серверна частина функціонуватиме за дотримання наступних мінімальних умов:

- CPU: Intel Core i5;
- RAM: 2 Gb;
- HDD/SSD: 1Gb (мінімально, для початкового запуску), але при збільшенні кількості користувачів, їх проєктів та налаштувань, для зберігання бази даних може знадобитися більший дисковий простір;
- WEB-сервер: Apache 2 або Nginx;
- база даних: MongoDB;
- порти: 80 (HTTP), 27018 (стандартний порт MongoDB);
- програмне забезпечення: середовище Node.js, менеджер пакетів NPM або Yarn.

Маючи посилання на репозиторій з вихідним кодом програміст може скопіювати його на локальний комп'ютер для подальшого доопрацювання за допомогою консольної команди `git clone «посилання_на_репозиторій»`, або візуального інтерфейсу, в залежності від програмного забезпечення для роботи з Git. Або ж завантаживши архів проєкту за посиланням.

Папка з колекціями документів бази даних повинна розміщуватись на одному сервері з серверною частиною застосування.

Щоб запустити серверну частину проєкту на локальному сервері для доопрацювання слід виконати наступні дії:

- за допомогою терміналу перейти до каталогу проєкту, в якому міститься файл `package.json`;
- ввести команду `npm start`;
- в терміналі з'явиться адреса порта, на якому працює потрібний сайт;
- необхідно ввести команду `sudo npm run compose` для підключення сервера до бази даних.

Для завантаження нових пакетів використовується команда `npm install «назва_пакету»`, для видалення — `npm uninstall «назва_пакету»`.

Скомпілювати проєкт для розміщення на хостингу можна командою `npm run build`, після чого всі необхідні файли з'являться в каталозі `dist`.

Зупиняється виконання проєкту на локальному сервері комбінацією `CTRL + C` при фокусі на терміналі.

Вхідними даними для застосування є колекції в базі даних, що зберігають інформацію про всі проєкти, дані користувачів, дані налаштувань облікових записів та проєктів, повідомлення.

Вихідними даними є візуальне представлення цих даних на клієнтській частині застосування. Дані між клієнтом і сервером передаються в тілі HTTP запитів/відповідей в вигляді строки формату JSON.

4.3 Тестування програми

Тестування програми було здійснено за декількома напрямками.

Перший напрямок включав перевірку реалізації функціональних вимог застосуванням. Для цього було розроблено набір тестів, що включали перевірку роботи запитів, що надсилаються до сервера, а також збереження та отримання даних з бази даних. Ці тести були розроблені на етапі написання самого програмного забезпечення.

Після цього була проведена окрема перевірка всіх функціональних вимог з метою підтвердження їх працездатності. Цей крок дозволяв переконатися, що програмне забезпечення відповідає всім заданим функціональним вимогам і працює належним чином.

Такий підхід до тестування гарантує, що функціональні вимоги програмного забезпечення виконуються правильно і результати його роботи відповідають очікуванням.

Для тестування серверної частини був використаний набір інструментів `Insomnia`.

`Insomnia` - це кросплатформовий клієнт для тестування API та взаємодії з вебсервісами. Він надає зручне середовище для відправлення

HTTP-запитів, отримання відповідей та налагодження вебзапитів. Insomnia дозволяє розробникам швидко і легко перевіряти працездатність API, тестувати різні запити, переглядати та аналізувати відповіді сервера [18].

Усі запити, реалізовані за допомогою Insomnia, надсилалися і оброблювалися серверною частиною коректно.

Приклади тестування показано на рисунках 4.6 – 4.7.

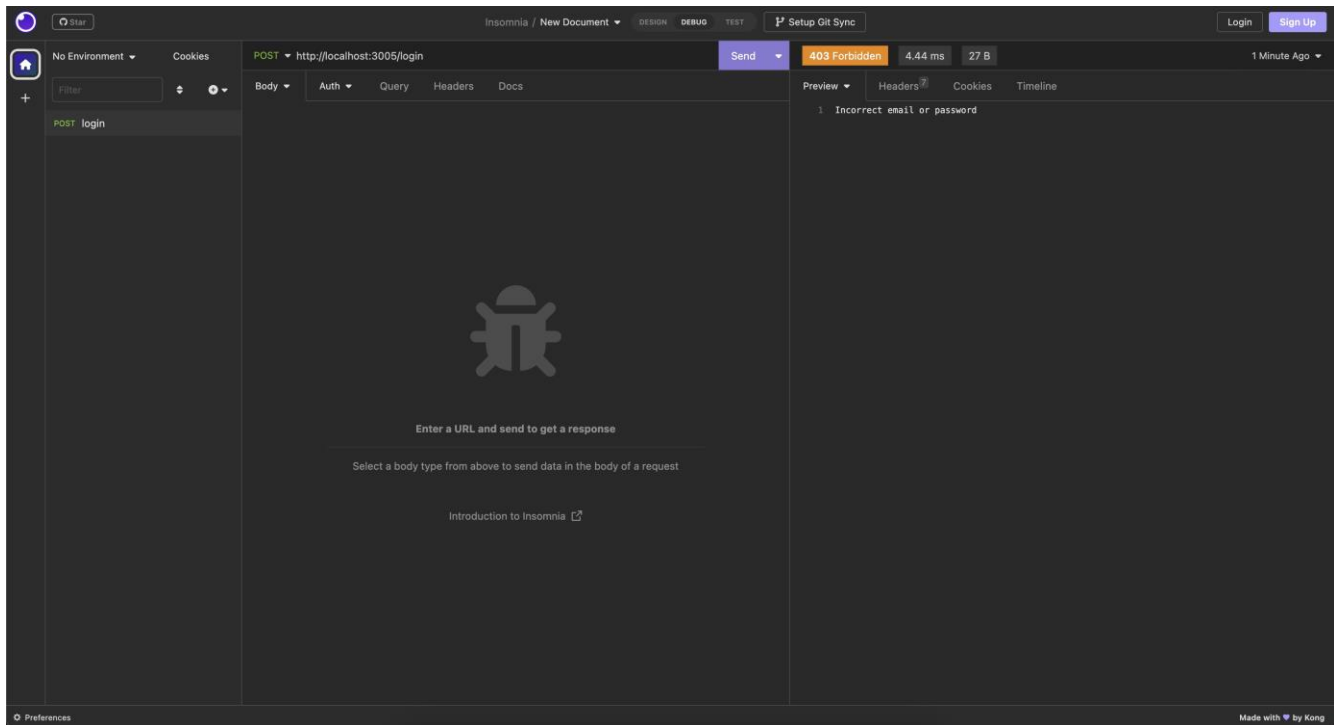


Рисунок 4.6 – Відповідь помилки під час авторизації

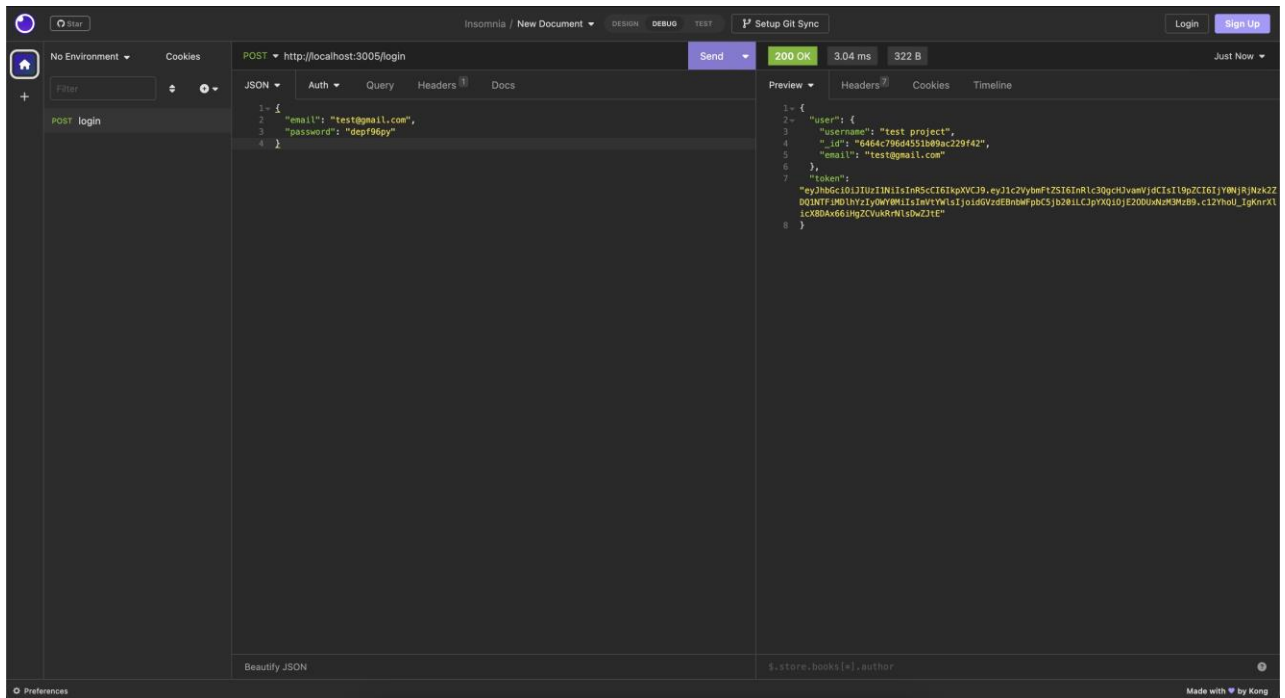


Рисунок 4.7 – Відповідь успіху під час авторизації

Другий напрямок включав тестування клієнтської частини вебзастосунку через різні браузер. Результати підтвердили, що програма є працездатною у всіх протестованих браузерах.

4.4 Висновки за розділом 4

Програмне забезпечення може виконувати операції збереження та отримання даних з бази даних, а також відправляти та отримувати HTTP-запити і відповіді. Виконано успішне тестування серверної частини вебзастосунку.

ВИСНОВКИ

У результаті виконання дипломної роботи була створена серверна частина програмного забезпечення, що спрямована на автоматизацію взаємодії між замовником та розробником програмного забезпечення.

Розроблене застосування має на меті поліпшити ефективність співпраці, зокрема швидкість та якість роботи, спростити взаємодію та розуміння між сторонами під час розробки програмного забезпечення. Це досягається шляхом автоматизації процесів, таких як оформлення проєктів, переговори, обробка даних, своєчасне уточнення деталей.

Серверна частина дозволяє керувати всіма проєктами замовника та розробника, переглядати, редагувати та видаляти дані проєкту, налаштовувати облікові записи для забезпечення максимальної зручності та ефективності роботи, відповідаючи на запити клієнтської частини.

Під час розробки було проведено аналіз предметної області, визначено ключові рішення щодо реалізації серверних компонентів системи та розроблено методику роботи з системою для замовника та розробника.

Серверна частина програмного забезпечення була розроблена у середовищі Visual Studio Code з використанням платформи Node.js. Програмне забезпечення розроблено таким чином, що воно може ефективно працювати під керуванням будь-якої операційної системи.

Розроблене програмне забезпечення демонструє гарну якість реалізації, надійність та ефективність в роботі. Інтерфейс програми є інтуїтивно зрозумілим і добре організованим, що сприяє зручності користування та швидкому освоєнню функціоналу.

Однією із переваг розробленого програмного забезпечення є його здатність автоматизувати процеси співпраці між замовником та розробником програмного забезпечення. Воно дозволяє замовнику з легкістю відстежувати процес робіт та спілкуватися з розробником. Розробнику ж надається

зручний інтерфейс для керування проєктами, редагування даних та спілкування з замовником.

Крім того, варто відзначити, що розроблене програмне забезпечення має гарну ступінь надійності та безпеки. Застосовані механізми перевірки та захисту даних, а також автентифікації користувачів забезпечують конфіденційність та цілісність інформації, що зберігається в системі.

Розроблене програмне ефективно виконує свої функції, полегшуючи роботу замовника та розробника, покращуючи якість співпраці.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Супровід програмного забезпечення. – [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%A1%D1%83%D0%BF%D1%80%D0%BE%D0%B2%D1%96%D0%B4_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F.
2. Документ "Web.pdf" Національного університету "Острозька академія". – [Електронний ресурс]. – Режим доступу: <http://www.ndu.edu.ua/liceum/web.pdf/>.
3. Smith, John. "Trello and Jira in Action: Benchmarking and Best Practices." Manning Publications, 2023, 372p.
4. Notion. "Notion Help & Support." Documentation. - [Electronic resource]. – Access mode: <https://www.notion.so/Help-Support-e040febf70fe4c04a36b487cdf6b5e7e>.
5. Стоянов, Іван. "Основи JavaScript: Програмування для фронтенду і бекенду." Apress, 2021, 464с.
6. Документ "7.pdf" журналу "Vernadsky National Library Scientific Herald". – [Електронний ресурс]. – Режим доступу: https://tech.vernadskyjournals.in.ua/journals/2018/6_2018/part_2/7.pdf.
7. Herron, David. "Node.js Web Development: Server-side Development with Node 10 made easy." Packt Publishing, 2018, 320с.
8. MongoDB. Documentation. - [Electronic resource]. – Access mode: <https://www.mongodb.com/>.
9. Документ "pract_sytnik_22.pdf" Київського національного економічного університету імені Вадима Гетьмана. – [Електронний ресурс]. – Режим доступу: https://ir.kneu.edu.ua/bitstream/handle/2010/38129/pract_sytnik_22.pdf?sequence

=1.

10. Mongoose. Documentation. - [Electronic resource]. – Access mode: <https://mongoosejs.com/>.

11. Pollock, James. "RESTful API Design: Best Practices in API Design with REST." Self-publishing, 2022, 300 p. Passport. Документація. – [Електронний ресурс]. – Режим доступу: <http://www.passportjs.org/>.

12. Mailgun. Documentation. - [Electronic resource]. – Access mode: <https://www.mailgun.com/>.

13. Busboy. Documentation. - [Electronic resource]. – Access mode: <https://www.npmjs.com/package/busboy>.

14. Cloudinary. Documentation. - [Electronic resource]. – Access mode: <https://cloudinary.com/documentation>.

15. Socket.io. Documentation. - [Electronic resource]. – Access mode: <https://socket.io/docs/>.

16. bcrypt. Documentation. - [Electronic resource]. – Access mode: <https://www.npmjs.com/package/bcrypt>.

17. Insomnia. Documentation. - [Electronic resource]. – Access mode: <https://support.insomnia.rest/>.

18. Mike Dirolf. Modeling data in MongoDB - [Electronic resource]. – Access mode: <https://m.habr.com/ru/post/144798/>.

19. А. Стягар. Серверна платформа NODE.JS – [Електронний ресурс]. – Режим доступу: http://elartau.tntu.edu.ua/bitstream/lib/23918/2/V-STC-IMST_2018_Stiahar_A-Node_js_platform_83.pdf.

20. MongoDB Online Guide - [Electronic resource]. – Access mode: <https://metanit.com/nosql/mongodb/>.

21. Brown, Ethan. "Web Development with Node and Express: Leveraging the JavaScript Stack." O'Reilly Media, 2018, 334c.

22. Pasquali, Sandro. "Mastering Node.js: Build robust and scalable real-time server-side web applications efficiently." Packt Publishing, 2017, 360c.

23. Clements, David Mark. "Node Cookbook: Over 50 recipes to master

the art of asynchronous server-side JavaScript using Node.js, 2nd Edition." Packt Publishing, 2018, 412c.

24. Richardson, Leonard, Amundsen, Mike. "RESTful Web APIs: Services for a Changing World." O'Reilly Media, 2013, 408c.

25. Banker, Kyle. "MongoDB in Action." Manning Publications, 2018, 560c.

26. Samuel Beckett. "Node.js Web Development: Server-side Web Development Made Easy." Packt Publishing, 2019, 542 p.

27. Jeremy Keith, Michael Nixon. "Designing Web APIs: Building APIs That Developers Love." O'Reilly Media, 2018, 298 p.

28. Christopher M. Barker. "MongoDB Applied Design Patterns: Practical Use Cases with the Leading NoSQL Database." Apress, 2013, 552 p.

29. Rick Grises. "Graph Databases: New Opportunities for Connected Data." O'Reilly Media, 2013, 238p.

30. Sean P. Merry. "Mastering MongoDB 4.x: Expert techniques to run high-volume and fault-tolerant database solutions using MongoDB." Packt Publishing, 2021, 748p.

ДОДАТОК А
Текст програми

src/index.js

```
import Koa from 'koa';
import config from './config/configuration';
import Router from 'koa-router';
import socket from 'socket.io';
import http from 'http';
import initPassport from './components/jwtStrategy';

import loginUser from './middlewares/loginUser';
import createUser from './middlewares/createUser';
import logoutUser from './middlewares/logoutUser';
import findUserEmail from './middlewares/findUserEmail';
import setUserPassword from './middlewares/setUserPassword';
import expireToken from './middlewares/expireToken';
import createProject from './middlewares/createProject';
import changePasswordSettings from './middlewares/changePasswordSettings';
import newSettings from './middlewares/newSettings';
import checkExpireToken from './middlewares/checkExpireToken';
import setAvatar from './middlewares/setAvatar';
import estimatorsProject from './middlewares/estimatorsProject';
import getUser from './middlewares/getUser';
import changeProjectStatus from './middlewares/changeProjectStatus';
import deleteProjectByld from './middlewares/deleteProject';
import deleteAvatar from './middlewares/deleteAvatar';
import findProjectByld from './middlewares/findProjectByld';
import changeProjectName from './middlewares/changeProjectName';
import changeProjectDescription from './middlewares/changeProjectDescription';
import changeClientName from './middlewares/changeClientName';
import changeClientEmail from './middlewares/changeClientEmail';
import createLibrarySection from './middlewares/createLibrarySection';
import setAttachment from './middlewares/setAttachment';
import deleteAttachment from './middlewares/deleteAttachment';
import createLibrarySectionItem from './middlewares/createLibrarySectionItem';
import getLibraries from './middlewares/getLibraries';
import createSection from './middlewares/createSection';
import createSectionItem from './middlewares/createSectionItem';
import editSection from './middlewares/editSection';
import deleteSection from './middlewares/deleteSection';
import editLibrarySection from './middlewares/editLibrarySection';
import deleteLibrarySection from './middlewares/deleteLibrarySection';
import fromLibrary from './middlewares/fromLibrary';
import toLibrary from './middlewares/toLibrary';
import getCustomer from './middlewares/getCustomer';
import customersProject from './middlewares/customersProject';
import getProjectSections from './middlewares/getProjectSection';
import editLibraryItem from './middlewares/editLibraryItem';
import deleteLibraryItem from './middlewares/deleteLibraryItem';
import editSectionItem from './middlewares/editSectionItem';
import deleteSectionItem from './middlewares/deleteSectionItem';
import updateOrderLibrary from './middlewares/updateOrderLibrary';
import updateOrderSection from './middlewares/updateOrderSection';
import editPriority from './middlewares/editPriority';
import downloadItemFromLibrary from './middlewares/downloadItemFromLibrary';
import editAdditional from './middlewares/editAdditional';
import getMessages from './middlewares/getMessages';
import updateMessage from './middlewares/updateMessage';
import deleteMessage from './middlewares/deleteMessage';

import { saveMessage } from './actions/saveMessage';
```

```

import bodyParser from 'koa-bodyparser';
import multer from 'koa-multer';

const cors = require('koa2-cors');

const app = new Koa();
const router = new Router();

const server = http.createServer(app.callback());
const io = socket(server);

server.listen(3005, () => {
  console.log('Application is starting on port 3000')
})

const upload = multer({
  storage: multer.memoryStorage()
});

app.use(async (ctx, next) => {
  try {
    await next();
  } catch (err) {
    ctx.status = err.status || 500;
    ctx.body = err.message;
  }
});

app.use(cors());
app.use(bodyParser({ multipart: true }));
app.use(router.routes());

app.use(initPassport());

router.post('/login', loginUser);

router.post('/create', createUser);

router.delete('/logout/:token', logoutUser);

router.get('/checkExpireToken', checkExpireToken);

router.post('/findemail', findUserEmail);

router.put('/setpassword', setUserPassword);

router.post('/expiretoken/:token', expireToken);

router.post('/createproject', createProject);

router.put('/changepasswordsettings', changePasswordSettings);

router.put('/settings', newSettings);

router.post('/setavatar', setAvatar);

router.get('/estimatorsproject/:token', estimatorsProject);

router.get('/getuser/:token', getUser);

```

```
router.put('/changeprojectstatus', changeProjectStatus);

router.delete('/deleteproject/:id', deleteProjectById);

router.delete('/deleteavatar/:token', deleteAvatar);

router.get('/findprojectbyid/:projectId', findProjectById);

router.put('/changeprojectname', changeProjectName);

router.put('/changeprojectdescription', changeProjectDescription);

router.put('/changeclientname', changeClientName);

router.put('/changeclientemail', changeClientEmail);

router.post('/setattachment', setAttachment);

router.delete('/deleteattachment/:attachmentId', deleteAttachment);

router.get('/getlibraries/:token', getLibraries);

router.post('/createlibrarysection', createLibrarySection);

router.post('/createlibrarysectionitem', createLibrarySectionItem);

router.put('/editlibrarysection', editLibrarySection);

router.delete('/deletelibrarysection/:sectionId', deleteLibrarySection);

router.get('/getprojectsections/:project', getProjectSections);

router.post('/createsection', createSection);

router.post('/createsectionitem', createSectionItem);

router.put('/editsection', editSection);

router.delete('/deletesection/:sectionId', deleteSection);

router.post('/fromlibrary', fromLibrary);

router.post('/tolibrary', toLibrary);

router.get('/getcustomer/:token', getCustomer);

router.get('/customersproject/:token', customersProject);

router.put('/editlibraryitem', editLibraryItem);

router.delete('/deletelibraryitem/:itemId', deleteLibraryItem);

router.put('/editsectionitem', editSectionItem);

router.delete('/deletesectionitem/:itemId', deleteSectionItem);

router.put('/updateorderlibrary', updateOrderLibrary);

router.put('/updateordersection', updateOrderSection);
```

```

router.put('/editpriority', editPriority);

router.post('/downloaditemfromlibrary', downloadItemFromLibrary);

router.put('/editadditional', editAdditional);

router.get('/getmessages/:projectId', getMessages);

router.put('/updatemessagetext', updateMessage);

router.delete('/deletemessage/:id', deleteMessage);

io.on('connection', (socket) => {

  socket.on('disconnect', () => {});

  let room = "";

  socket.on('room', (data) => {
    room = data;
    socket.join(room);
  });

  socket.on('message', (data) => {
    socket.broadcast.to(room).emit('messages', 'New message');
    saveMessage(data);
  })

});

src/actions/getAllEstimatorProject.js
import Token from '../model/tokenModel';
import estimatorModel from '../model/estimatorModel';

const getAllEstimatorProject = async (token) => {
  const userToken = await Token.findOne({ accessToken: token });
  const { userId } = userToken;
  const estimatorId = userId;

  const estimatorsProject = await estimatorModel.aggregate([
    {
      $match: { userId: estimatorId },
    },
    {
      $lookup: {
        from: "projects",

        let: { estimatorProject: "$projectId" },
        pipeline: [
          {
            $match:
            {
              $expr:
              {
                $and:
                [
                  { $eq: ["$_id", "$$estimatorProject"] },
                ]
              }
            }
          }
        ]
      }
    }
  ])
};

```

```

    }
  }
},
{ $project: { _id: 0, __v: 0 } }
],
as: "estimator_project",
},
},
{ $project: { _id: 0, userId: 0, __v: 0, } },

{
  $lookup: {
    from: "clients",

    let: { estimatorProject: "$projectId" },
    pipeline: [
      {
        $match:
        {
          $expr:
          {
            $and:
            [
              { $eq: ["$projectId", "$$estimatorProject"] },
            ]
          }
        }
      },
      { $project: { _id: 0, __v: 0, } }
    ],
    as: "estimator_client",
  },
},

{ $project: {
  projectId: 1,
  estimator_project: 1,
  client: { $arrayElemAt: ["$estimator_client", 0] },
} },

{
  $lookup: {
    from: "users",

    let: { estimatorClientId: "$client.userId" },
    pipeline: [
      {
        $match:
        {
          $expr:
          {
            $and:
            [
              { $eq: ["$_id", "$$estimatorClientId"] },
            ]
          }
        }
      },
      { $project: { _id: 0, __v: 0, password: 0, role: 0 } }
    ],
  },
}

```

```

    ],
    as: "client_user",
  },
},
{
  $lookup: {
    from: "additional",

    let: { estimatorProject: "$projectId" },
    pipeline: [
      {
        $match:
        {
          $expr:
          {
            $and:
            [
              { $eq: ["$projectId", "$$estimatorProject"] },
            ]
          }
        }
      },
      { $project: { _id: 0, __v: 0, projectId: 0 } }
    ],
    as: "additional",
  },
},
{
  $lookup: {
    from: "attachments",
    let: { estimatorProject: "$projectId" },
    pipeline: [
      {
        $match:
        {
          $expr:
          {
            $and:
            [
              { $eq: ["$projectId", "$$estimatorProject"] },
            ]
          }
        }
      },
      { $project: { __v: 0, projectId: 0, type: 0 } }
    ],
    as: "attachment",
  },
},
{
  $lookup: {
    from: "sections",
    let: { estimatorProject: "$projectId" },
    pipeline: [
      {
        $match:
        {
          $expr:

```

```

        {
          $and:
            [
              { $eq: ["$projectId", "$$estimatorProject"] },
            ]
        }
      ],
    },
    {
      $lookup: {
        from: "tasks",
        let: { section_id: "$_id" },
        pipeline: [
          {
            $match:
              {
                $expr:
                  {
                    $and:
                      [
                        { $eq: ["$sectionId", "$$section_id"] },
                      ]
                  }
              },
          ],
          { $project: { __v: 0, projectId: 0, type: 0 } }
        ],
        as: "task",
      },
    },
    { $project: { __v: 0, projectId: 0, type: 0 } }
  ],
  as: "section",
},
{
  $project: {
    projectId: 1,
    projects: { $arrayElemAt: ["$estimator_project", 0] },
    users: { $arrayElemAt: ["$client_user", 0] },
    additional: { $arrayElemAt: ["$additional", 0] },
    attachment: 1,
    section: 1,
  }
},
]);

return estimatorsProject;
}

export default getAllEstimatorProject;

src/actions/emailUtilsForProjects.js
import config from '../config/configuration';
const emailConfig = require('../config/emailConfig')();
const mailgun = require('mailgun-js')(emailConfig);

exports.sendProjectEmail = (email, projectId, token) =>
  new Promise((resolve, reject) => {

```

```

const data = {
  from: 'yevheniiukraintseva@gmail.com',
  to: email,
  subject: 'Estimatify. Create Project',
  text: [
    `Link to your project: ${process.env.FRONT_URL}/customerpage/project/${projectId}/${token}`,
    `Link to your dashboard: ${process.env.FRONT_URL}/customerpage?token=${token}`
  ]
};

mailgun.messages().send(data, (error) => {
  if (error) {
    return reject(error);
  }
  return resolve();
});
});

src/middlewares/setAvatar.js
import Busboy from 'busboy';
import path from 'path';
import os from 'os';
import fs from 'fs';
import * as _ from 'lodash';
import util from 'util';

import publicFields from '../components/db/commonFields/user';
import findUser from '../actions/findUserWithToken';
import updateUserAvatar from '../actions/updateUserAvatar';

const cloudinary = require('cloudinary').v2;

cloudinary.config({
  cloud_name: 'dnlsyw0df',
  api_key: '791224125289934',
  api_secret: 'VdJSIqpoQPxbmQNoljMjzPBMPc8'
});

const setAvatar = async (ctx) => {
  try {
    const token = ctx.request.headers.authorization;
    const user = await findUser(token);
    let myImage;

    let busboy = new Busboy({ headers: ctx.request.headers });
    busboy.on('file', function (fieldname, file, filename, encoding, mimetype) {
      file.on('data', function (data) {
      });
      file.on('end', function () {
        myImage = {
          fieldname,
          filename,
          encoding,
          mimetype,
        };
      });
    });
    let saveTo = path.join( os.tmpdir(), path.basename(user._id.toString()));
    file.pipe(fs.createWriteStream(saveTo));
  });
  busboy.on('field', function (fieldname, val, fieldnameTruncated, valTruncated, encoding, mimetype) {

```

```

});

ctx.req.pipe(busboy);

const User = await new Promise( (resolve) => {

  busboy.on('finish', async function () {
    const { filename } = myImage;

    if (!user) {
      throw({
        status: 500,
        message: 'User not found',
      })
    }

    if(user.avatar_id) {
      const destroy = util.promisify(cloudinary.uploader.destroy);
      const res = await destroy(user.avatar_id)
        .catch((error) => {
          throw(error)
        });
    }

    const upload = util.promisify(cloudinary.uploader.upload);
    const image = await upload('/tmp/' + user._id, { tags: filename })
    if (image) {
      fs.unlinkSync('/tmp/' + user._id);

      const updatedAvatar = await updateUserAvatar(user._id, image.public_id, image.secure_url);
      if (!updatedAvatar) {
        throw({
          status: 500,
          message: 'Avatar is not updated',
        })
      }

      const getUser = await findUser(token);
      if (!getUser) {
        throw({
          status: 404,
          message: 'User not found'
        })
      }

      const result = _.pick(getUser, publicFields);
      resolve(result);

    } else {
      throw({
        status: 500,
        message: 'Image is not uploaded to Cloudinary'
      })
    }
  });
});

ctx.status = 200;
ctx.body = User;
return;

```

```

    } catch (error) {
      throw(error)
    }
  }
}

export default setAvatar;

src/middlewares/deleteAvatar.js
import util from 'util';

import User from '../model/userModel';

import findUser from '../actions/findUserWithToken';
import { deleteUserAvatar } from '../actions/deleteUserAvatar';

const cloudinary = require('cloudinary').v2;

cloudinary.config({
  cloud_name: 'dnlsyw0df',
  api_key: '791224125289934',
  api_secret: 'VdJSIqpoQPxbmQNoljMjzPBMPc8'
});

const deleteAvatar = async (ctx) => {
  try {
    const user = await findUser(ctx.params.token)

    const destroy = util.promisify(cloudinary.uploader.destroy);
    await destroy(user.avatar_id)
      .catch((error) => {
        throw (error)
      });

    await deleteUserAvatar(user._id);

    ctx.status = 200;

  } catch (error) {
    throw (error);
  }
}

export default deleteAvatar;

src/middlewares/createProject.js
import jwt from 'jsonwebtoken';
import generator from 'generate-password';

import projectModel from '../model/projectModel';
import estimatorModel from '../model/estimatorModel';
import Token from '../model/tokenModel';
import User from '../model/userModel';
import clientModel from '../model/clientModel';
import additionalModel from '../model/additionalModel';

import { sendProjectEmail } from '../actions/emailUtilsForProjects';
import { setClientToken } from '../actions/actionClientToken';

```

```

const createProject = async (ctx) => {

  const { settings } = ctx.request.body;

  const { clientEmail, name, projectName, description, token } = ctx.request.body;

  const userToken = await Token.findOne({ accessToken: token });
  const client = await User.findOne({ email: clientEmail });

  if (!client) {

    let userPasssword = await generator.generate({
      length: 8,
      numbers: true,
      uppercase: false,
    });

    const findUsername = await User.findOne({ username: name });
    if (!findUsername) {
      await User.create({ username: name, email: clientEmail, password: userPasssword });
    } else {
      throw ({
        status: 409,
        message: 'User with this username is already exists'
      })
    }
  }

  } else if (client.role == 'Estimator') {
    throw ({
      status: 412,
      message: 'Client cannot be the Estimator',
    })
  }

  const projectsClient = await User.findOne({ email: clientEmail });
  if (projectsClient) {

    const project = await projectModel.create({ name: projectName, descriptions: description });

    await clientModel.create({ userId: projectsClient._id, projectId: project._id });
    await estimatorModel.create({ userId: userToken.userId, projectId: project._id });

    await additionalModel.create({
      projectId: project._id,
      bugfix: settings.bugfix,
      codereview: settings.codereview,
      test: settings.test,
      pen: settings.pen,
    });

    const token = await jwt.sign({ projectsClient }, process.env.SECRET_CLIENT);
    setClientToken(projectsClient._id, token);
    sendProjectEmail(projectsClient.email, project._id, token);

    ctx.body = project;

  }
}

export default createProject;

```

```

src/middlewares/expireToken.js
import { existsToken } from '../actions/existsToken';

const expireToken = async (ctx) => {
  try {
    const token = await existsToken(ctx.params.token);
    if (!token) {
      ctx.body = { error: 'Invalid link. Not found' };
      return;
    }

    if (new Date(token.expireAt).getTime() < new Date().getTime()) {
      ctx.body = { error: 'Link is expired. Reset password again' };
      return;
    }

  } catch (error) {
    ctx.body = { error: 'Invalid link' };
    return;
  }
  return ctx.status = 200;
}

export default expireToken;

src/components/jwtStrategy.js
import passport from 'passport';
import jwt from 'jsonwebtoken';
import User from '../model/userModel';
import Token from '../model/tokenModel';
import * as _ from 'lodash';
import publicFields from '../components/db/commonFields/user';

import { Strategy } from 'passport-jwt';
import { ExtractJwt } from 'passport-jwt';

const JwtStrategy = Strategy;

export default function () {
  const opts = {}
  opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
  opts.secretOrKey = process.env.SECRET;

  passport.use(new JwtStrategy(opts, async (jwt_payload, done) => {
    const tokenRec = await Token.findOne({
      accessToken: jwt.sign(jwt_payload, process.env.SECRET),
      expireAt: {
        $gt: new Date(),
      }
    });

    if (!tokenRec) {
      return done(null, null);
    }

    User.findOne({ _id: jwt_payload._id }, (err, user) => {
      if (err) {
        return done(err, null);
      }
    })
  })
}

```

```

    if (user) {
      return done(null, _.pick(user, publicFields));
    }
    else {
      return done(err, null);
    }
  });
}));

return passport.initialize()
}

```

```

src/db/additionalSchema.js
import { Schema } from 'mongoose';

```

```

const additionalSchema = new Schema({
  projectId: {
    type: Schema.Types.ObjectId,
    required: true,
    index: true
  },

  bugfix: {
    type: Number,
  },

  codereview: {
    type: Number,
  },

  test: {
    type: Number,
  },

  pen: {
    type: Number,
  },
});

export default additionalSchema;

```

```

src/db/attachmentSchema.js
import { Schema } from 'mongoose';

```

```

const attachmentSchema = new Schema({
  projectId: {
    type: Schema.Types.ObjectId,
    required: true,
    index: true
  },
  fileName: {
    type: String,
    required: true,
  },
  type: {
    type: String,
    required: true,
  },
  url: {
    type: String,

```

```

      required: true,
    },
    attachment_id: {
      type: String,
    }
  });

export default attachmentSchema;

src/db/clientSchema.js
import { Schema, SchemaTypes } from 'mongoose';

const clientSchema = new Schema({
  userId: {
    type: Schema.Types.ObjectId,
    required: true,
    index: true,
  },

  projectId: {
    type: Schema.Types.ObjectId,
    required: true,
    index: true,
  },
});

export default clientSchema;

src/db/clientToken.js
import { Schema, SchemaTypes } from 'mongoose';

const clientTokenSchema = new Schema({
  userId: {
    type: Schema.Types.ObjectId,
    required: true,
    index: true
  },

  accessToken: {
    type: String,
    required: true,
    index: true
  },
});

export default clientTokenSchema;

src/db/emailTokenSchema.js
import { Schema, SchemaTypes } from 'mongoose';

import moment from 'moment';

const emailTokenSchema = new Schema({
  userId: {
    type: Schema.Types.ObjectId,
    required: true,
    index: true,
  },
});

```

```

    accessToken: {
      type: String,
      required: true,
      index: true,
    },

    expireAt: {
      type: Date,
      required: true,
      default: new Date(moment().add(30, 'minutes')),
    }
  })

```

```
export default emailTokenSchema;
```

```

src/db/estimatorSchema.js
import { Schema, SchemaTypes } from 'mongoose';

```

```

const estimatorSchema = new Schema({
  userId: {
    type: Schema.Types.ObjectId,
    required: true,
    index: true,
  },

  projectId: {
    type: Schema.Types.ObjectId,
    required: true,
    index: true,
  },
})

```

```
export default estimatorSchema;
```

```

src/db/libraryItem.js
import { Schema } from 'mongoose';

```

```

const libraryItemSchema = new Schema({
  librarySectionId: {
    type: Schema.Types.ObjectId,
    required: true,
    index: true
  },
  order: {
    type: Number,
    required: true,
  },
  name: {
    type: String,
    required: true,
  },
  optimistic: {
    type: Number,
    required: true,
  },
  likely: {
    type: Number,
    required: true,
  },
  pessimistic: {

```

```

      type: Number,
      required: true,
    },
    average: {
      type: Number,
      required: true,
    }
  }
});

```

```
export default libraryItemSchema;
```

```
src/db/librarySchema.js
import { Schema } from 'mongoose';
```

```
const librarySchema = new Schema({
  userId: {
    type: Schema.Types.ObjectId,
    required: true,
  }
});
```

```
export default librarySchema;
```

```
src/db/librarySectionSchema.js
import { Schema } from 'mongoose';
```

```
const librarySectionSchema = new Schema({
  libraryId: {
    type: Schema.Types.ObjectId,
    required: true,
    index: true
  },
  name: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  }
});
```

```
export default librarySectionSchema;
```

```
src/db/messageSchema.js
import { Schema } from 'mongoose';
```

```
const messageSchema = new Schema({
  projectId: {
    type: Schema.Types.ObjectId,
    required: true,
  },
  ownerId: {
    type: Schema.Types.ObjectId,
    required: true,
  },
  text: {
    type: String,
    required: true,
  },
});
```

```

    createdAt: {
      type: Date,
      required: true,
      default: new Date(),
    },
    updatedAt: {
      type: Date,
      required: true,
      default: new Date(),
    }
  });

```

```
export default messageSchema;
```

```
src/db/projectSchema.js
import { Schema } from 'mongoose';
```

```

const projectSchema = new Schema({
  name: {
    type: String,
    required: true,
  },
  status: {
    type: String,
    enum: ['Active', 'Archived'],
    default: 'Active',
  },
  descriptions: {
    type: String,
  },
  date: {
    type: Date,
    required: true,
    default: new Date(),
  },
  updatedAt: {
    type: Date,
    default: new Date(),
  }
});

```

```
export default projectSchema;
```

```
src/db/sectionSchema.js
import { Schema } from 'mongoose';
```

```

const sectionSchema = new Schema({
  projectId: {
    type: Schema.Types.ObjectId,
    required: true,
  },
  name: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  }
});

```

```
export default sectionSchema;
```

```
src/db/settingsSchema.js
```

```
import { Schema } from 'mongoose';
```

```
const settingsSchema = new Schema({
  userId: {
    type: Schema.Types.ObjectId,
    required: true,
    index: true
  },

  bugfix: {
    type: Number,
    default: 10,
  },

  codereview: {
    type: Number,
    default: 10,
  },

  test: {
    type: Number,
    default: 10,
  },

  pen: {
    type: Number,
    default: 10,
  },
  updatedAt: {
    type: Date,
    required: true,
    default: new Date(),
  }
});
```

```
export default settingsSchema;
```

```
src/db/taskSchema.js
```

```
import { Schema } from 'mongoose';
```

```
const taskSchema = new Schema({
  sectionId: {
    type: Schema.Types.ObjectId,
    required: true,
  },
  priority: {
    type: String,
    enum: ['High', 'Medium', 'Low'],
    default: 'Medium',
  },
  order: {
    type: Number,
    required: true,
  },
  name: {
    type: String,
```

```

      required: true,
    },
    optimistic: {
      type: Number,
      required: true,
    },
    likely: {
      type: Number,
      required: true,
    },
    pessimistic: {
      type: Number,
      required: true,
    },
    average: {
      type: Number,
      required: true,
    }
  }
});

export default taskSchema;

src/db/userSchema.js
import { Schema } from 'mongoose';

const userSchema = new Schema({
  username: {
    type: String,
    required: true,
    index: true,
    unique: true
  },
  email: {
    type: String,
    required: true,
    index: true,
    unique: true
  },
  password: {
    type: String,
    required: true
  },
  avatar: String,
  avatar_id: {
    type: String,
  },
  role: {
    type: String,
    enum: ['Client', 'Estimator'],
    default: 'Client',
  },
  createdAt: {
    type: Date,
    required: true,
    default: new Date(),
  },
  updatedAt: {
    type: Date,
    required: true,
    default: new Date(),
  },
});

```

```

    }
  });

  export default userSchema;

src/model/userModel.js
import mongoose from '../components/db';
import userSchema from '../db/userSchema';

export default mongoose.model('User', userSchema);

src/validator/access.js
import * as _ from 'lodash';
import User from '../model/userModel';
import publicFields from '../components/db/commonFields/user';

class AccessValidator {
  async login(body) {

    const { email, password } = body;
    const user = await User.findOne({ email });

    if (user && email === user.email && password === user.password) {
      return _.pick(user, publicFields);
    } else {
      throw new Error('Access denied!');
    }
  }
}

export default AccessValidator;
export const accessValidator = new AccessValidator();

src/validator/passwordEmail.js
import validator from 'validator';
import passwordValidator from 'password-validator';

var schema = new passwordValidator();

schema.is().min(5)

const passwValidator = (body, next) => {
  const {email, password} = body;
  if((validator.isEmail(email)) && (schema.validate(password))) {
    return true;
  } else {
    console.log('Invalid password')
    throw new Error('Invalid password');
  }
}

export default passwValidator;

```

ДОДАТОК Б
Слайди презентації

Національний університет «Запорізька політехніка»

Розробка серверної частини вебзастосунку для автоматизації роботи між замовником і розробником програмного забезпечення

Автор:

студентка групи КНТ-110 сп Українцева Є. С.

Керівник:

к.т.н., доцент, Каплієнко Т. І.

Рисунок Б.1 – Слайд № 1

2

Мета, об'єкт, предмет і задачі дослідження

- Мета - дослідження розвитку сучасних серверних вебтехнологій та їх ефективного застосування на прикладі створення серверної частини вебсайту, призначеного для автоматизації роботи з проектом під час його створення і реалізації між замовником і розробником програмного забезпечення.
- Об'єкт дослідження – програмні засоби для автоматизації взаємодії між замовником і розробником програмного забезпечення.
- Предмет дослідження – серверна частина вебзастосунку для автоматизації та покращення ефективності взаємодії між замовником і розробником програмного забезпечення.
- У даній роботі було поставлено завдання створити серверну частину програмного забезпечення для автоматизації роботи між замовником і розробником програмного забезпечення.

Рисунок Б.2 – Слайд № 2

3

Аналіз предметної області

- Визначено зміст сутностей: розробник програмного забезпечення, процес розробки програмного забезпечення, програмне забезпечення, замовник, сайт.
- Проаналізовано класифікації сайтів відповідно до доступних сервісів, фізичного розташування, схеми представлення інформації.
- Створений сайт має задовольняти критерії зовнішнього закритого інтернет-представництва власників бізнесу.

Рисунок Б.3 – Слайд № 3

4

Огляд програмних аналогів

Система управління проєктами	Опис	Переваги	Масштаби команди
Trello	Програмне забезпечення для керування проєктами з використанням дошок Kanban.	- Простий та інтуїтивно зрозумілий інтерфейс; - Універсальний пакет програмного забезпечення, який підходить для більшості підприємств; - Гнучкість у використанні та адаптабельність до різних типів проєктів.	Різні типи проєктів, різний розмір команди.
Jira	Програмне забезпечення для керування проєктами з різними типами дошок, включаючи Kanban, Scrum і спринт.	- Розширені функції, включаючи відстеження помилок і можливість налаштування програмного забезпечення; - Врахування проєктів у різних випусках. - Підходить для традиційних команд розробників ПЗ.	Традиційні команди розробників ПЗ.
Notion	Система керування файлами на основі шаблонів з різноманітними функціями, включаючи часові шкали, діаграми Ганта, бюджетування тощо.	- Динамічний спосіб управління та розробки вмісту; - Візуалізація роботи над проєктом за допомогою наочних показників; - Підходить для компаній, яким потрібен ресурсний центр та злиття команд та груп.	Компанії з акцентом на управління вмістом.

Рисунок Б.4 – Слайд № 4

Вибір мови програмування серверної частини застосування

5

Особливості	JavaScript	Python
Популярність	Широко використовується на клієнтській стороні та сервері за допомогою Node.js	Популярність як мови розробки вебзастосунків
Асинхронність	Підтримується з використанням промісів та асинхронних функцій (async/await)	Підтримується з використанням бібліотек, таких як asyncio
Швидкодія	Швидкість виконання може залежати від оптимізації коду та використання розширень	Зазвичай має високу швидкодію, особливо у комбінації з популярними фреймворками
Екосистема	Велика кількість пакетів та бібліотек на npm	Велика кількість пакетів та бібліотек на PyPI

Рисунок Б.5 – Слайд № 5

Структурна схема сервера

6

Структурно серверна частина складається з наступних модулів:

- actions – описані основні події;
- components – описані основні компоненти застосування;
- config – містяться компоненти конфігурації ПЗ;
- middlewares – описані обробники даних;
- model – міститься опис моделей БД;
- validator – описані функції валідації даних перед і після роботи з БД.

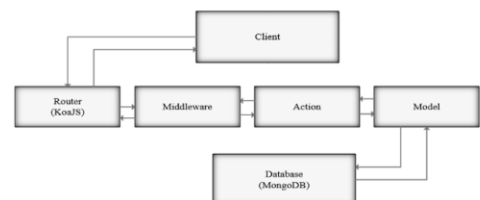


Рисунок Б.6 – Слайд № 6



Рисунок Б.7 – Слайд № 7

8

Вибір бази даних

БД	Переваги	Недоліки
SQL	- атомарність, узгодженість, ізолюваність, довговічність; - таблиця містить одну або декілька категорій даних у стовпцях; - рядок містить унікальний екземпляр даних для категорій, визначених стовпцями; - користувач може мати доступ до даних з бази даних, не знаючи структури таблиці.	- необхідність у масштабуванні на потужних серверах; - необхідність ускладнювати систему, що робить обробку більш важкою; - дорогі системи зберігання та запатентованих серверів.
NoSQL	- зберігання погано структурованих великих обсягів даних; - швидка обробка даних; - значно дешевші в обслуговуванні, ніж реляційні; - краща масштабованість	- узгодженість даних, ризики синхронізації вузлів; - відсутність стандартизації; - погано розповсюджуються на декілька вузлів.
MongoDB	- відсутність схеми: дана БД заснована на колекціях різних документів. Кількість полів, зміст і розмір цих документів може відрізнятися; - зрозуміла структура кожного об'єкта; - легко масштабується; - зберігання використовуваних в даний момент даних здійснюється за допомогою внутрішньої пам'яті, що дозволяє отримувати більш швидкий доступ; - дані зберігаються у вигляді JSON документів; - MongoDB підтримує динамічні запити документів (document-based query); - відсутність складних JOIN запитів.	- немалий розмір даних; - відсутність звичного JOIN запиту; - недосконала система Map/Reduce.

Рисунок Б.8 – Слайд № 8

Спроектowana об'єктно-орієнтована модель

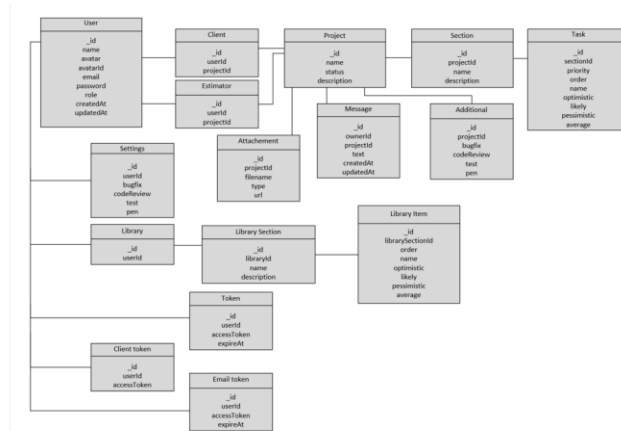


Рисунок Б.9 – Слайд № 9

Алгоритм роботи токєну

Один з обов'язкових алгоритмів взаємодії між клієнтською і серверною частинами полягає у перевірці валідності токєна, включаючи його термін дії. Ця перевірка здійснюється перед кожним запитом до сервера і забезпечує безпеку. Для цього використовується JWT (JSON Web Token) технологія, з використанням функції `passport.authenticate`.

```

import passport from 'passport';
import Jwt from 'jsonwebtoken';
import User from '../model/userModel';
import Token from '../model/tokenModel';
import ms from 'ms';
import publicFields from '../components/db/commonFields/user';

import { Strategy } from 'passport-jwt';
import { ExtractJwt } from 'passport-jwt';

const JwtStrategy = Strategy;

export default function () {
  const opts = {}
  opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
  opts.secretOrKey = process.env.SECRET;

  passport.use(new JwtStrategy(opts, async (jwt_payload, done) => {
    const tokenRec = await Token.findOne({
      accessToken: jwt.sign(jwt_payload, process.env.SECRET),
      expiresAt: {
        $gt: new Date(),
      },
    });

    if (!tokenRec) {
      await Token.deleteMany({
        expiresAt: {
          $lt: new Date(),
        },
      });
      return done(null, null);
    }

    User.findOne({ _id: jwt_payload._id }, (err, user) => {
      if (err) {
        return done(err, null);
      }
      if (user) {
        return done(null, __pick(user, publicFields));
      }
      else {
        return done(err, null);
      }
    });
  }));

  return passport.initialize();
}

```

Рисунок Б.10 – Слайд № 10

11

Алгоритм створення проєкту

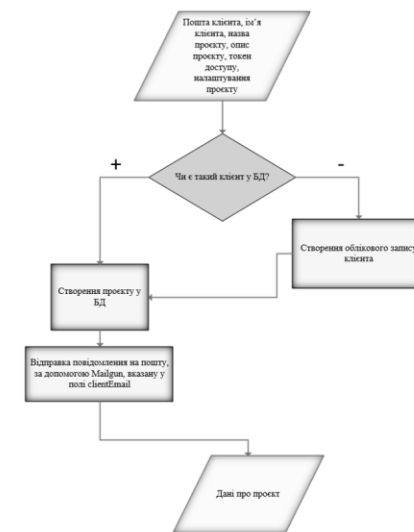


Рисунок Б.11 – Слайд № 11

12

Алгоритм роботи чату

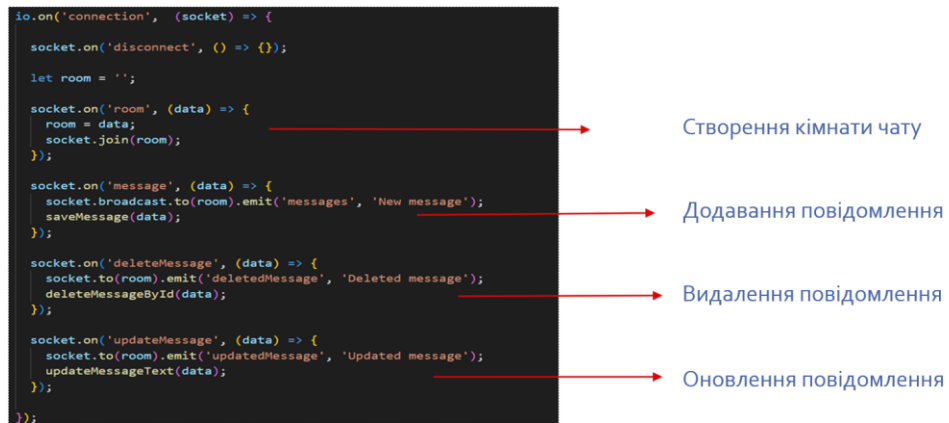


Рисунок Б.12 – Слайд № 12

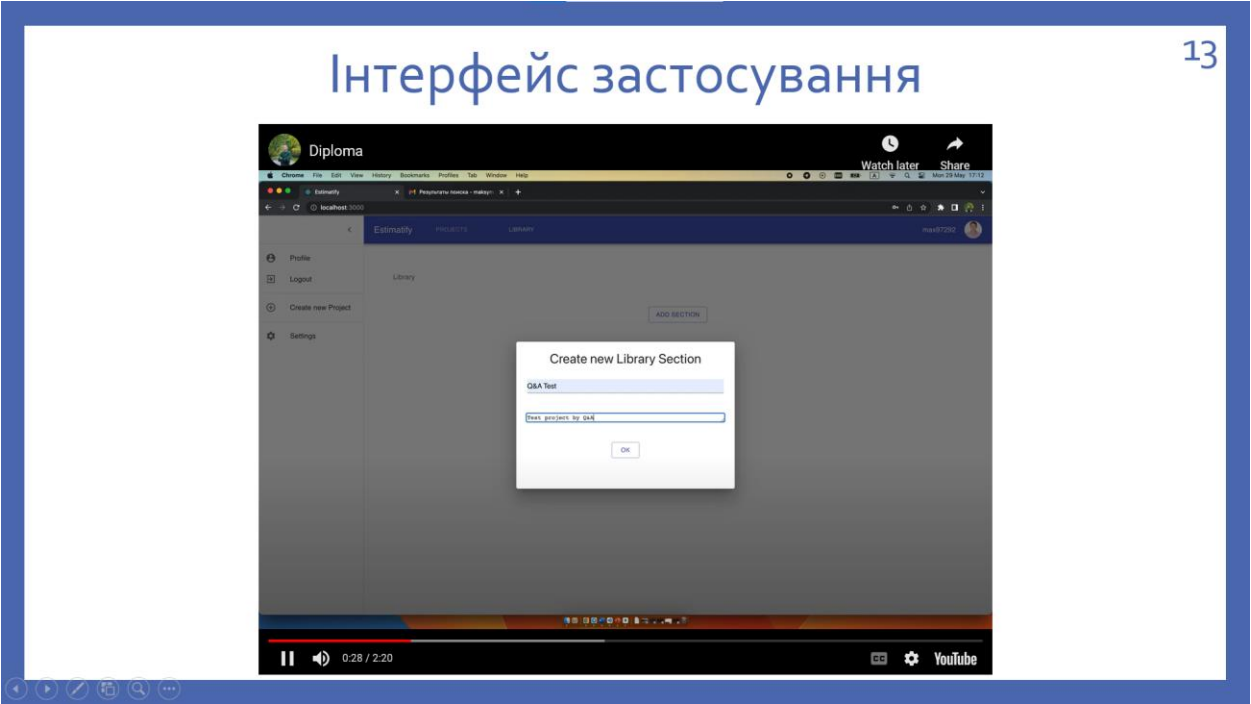
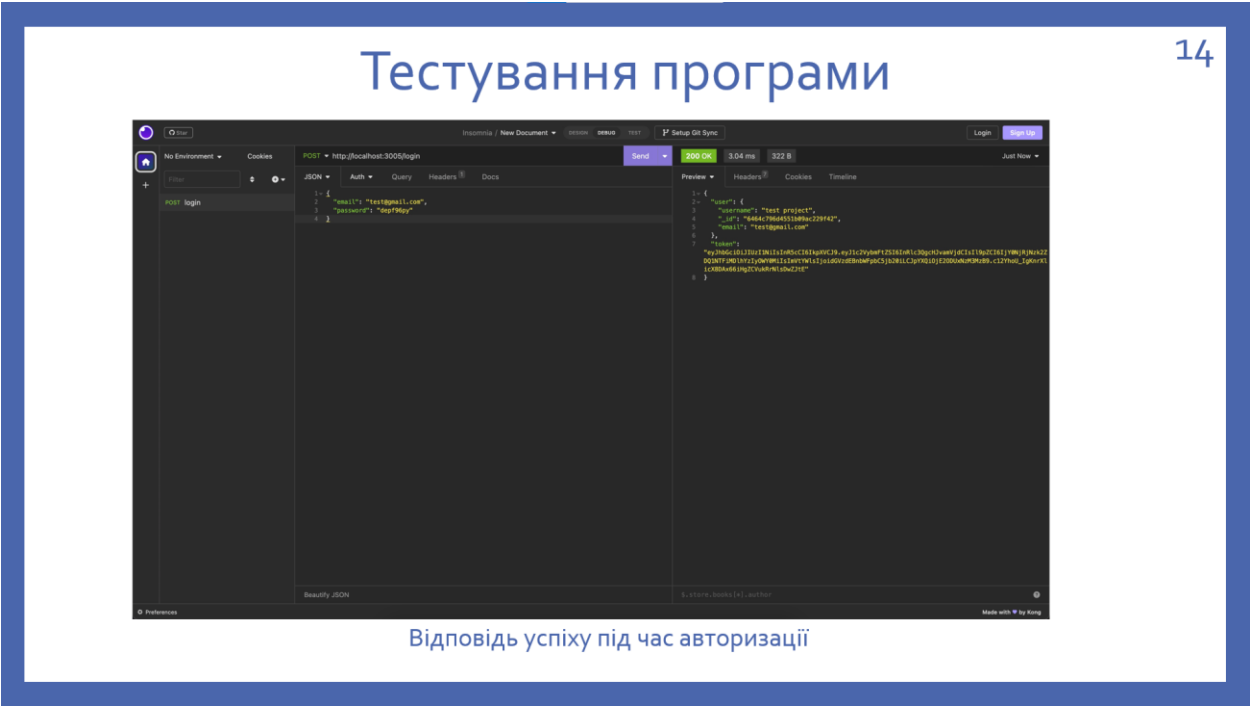


Рисунок Б.13 – Слайд № 13



Відповідь успіху під час авторизації

Рисунок Б.14 – Слайд № 14

ВИСНОВКИ

- Розроблене програмне забезпечення демонструє гарну якість реалізації, надійність та ефективність в роботі.
- Інтерфейс є інтуїтивно зрозумілим і добре організованим, що сприяє зручності користування та швидкому освоєнню функціоналу.
- Програмне забезпечення має гарну ступінь надійності та безпеки.
- Забезпечення здатне автоматизувати процеси співпраці між замовником та розробником програмного забезпечення, а також забезпечувати своєчасну взаємодію.
- Розроблений застосунок не має альтернатив на сучасному ринку українських програмних продуктів, поєднує кращі методи аналогів та доповнений власними принципами, заснованими на спостереженнях.

Рисунок Б.15 – Слайд № 15