Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра мультимедійних систем

Аналіз та розробка програм для керування робочими процесами командної розробки

Текстова частина до курсової роботи за спеціальністю 121 «Інженерія програмного забезпечення»

Керівник курсової роботи
ст.в.___Салата К.В. ___

(прізвище та ініціали)

_____

(підпис)

“____” _____ 2022 р.

Виконав студент Євтушенко І.О.

(прізвище та ініціали)

_____

(підпис)

“____” _____ 2022 р.

Київ-2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Завідувач кафедри мультимедійних систем,

канд. фіз-мат. наук, доц. – Гороховський С.С.

_____ (підпис)

“____” _____ 2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Євтушенку Ігорю Олеговичу

Факультету інформатики 4 р.н. бакалаврської програми

ТЕМА: Аналіз та розробка програм для керування робочими процесами командної розробки

Зміст ТЧ до курсової роботи:

Календарний план

Вступ

Огляд теоретичного матеріалу та здійснення дослідження

Висновки

Список використаної літератури та посилань

Додатки (за необхідністю)

Дата видачі “____” _____ 2022 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Тема: Аналіз та розробка програм для керування робочими процесами командної розробки

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	01.12.2021	
2.	Пошук тематичної літератури	15.12.2021	
3.	Ознайомлення з літературою	01.02.2022	
4.	Вивчення аналогів	20.04.2022	
5.	Проектування бази даних	20.04.2022	
6.	Створення UI частини застосунку	01.05.2022	
7.	Проектування та розробка серверної частини	01.05.2022	
8.	Проектування та розробка веб-частини застосунку	01.05.2022	
9.	Написання текстової частини	15.05.2022	
10.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	18.05.2022	
11.	Захист роботи	20.05.2022	

ЗМІСТ

АНОТАЦІЯ

У курсовій роботі розглянуті способи розробки веб-застосунку, використовуючи бібліотеку React та фреймворку для роботи з серверними застосунками Node.js. Більша частина уваги була зосереджена на поетапній побудові застосунку, проектуванні дизайну бази даних, дотримуючись правил нормалізації, та на деталях реалізації самого застосунку. Практична частина курсової роботи базується на реалізації самої програми, а саме застосунку для керування робочими процесами командної роботи.

У першій частині розглянута предметна область застосунку та проведена аналітика із застосунками-конкурентами.

У другій частині розглянуті теоретичні відомості про стек технологій, яким розроблявся застосунок, а також розглядається авторизація та аутентифікація.

У третій частині розглядається практична частина курсової роботи. Описуються реалізації як серверної частини та і клієнтської. Також підводяться підсумки та результати всієї роботи.

ВИКОРИСТАННЯ СКОРОЧЕНЬ

UI - (user interface), у перекладі “інтерфейс користувача”.

React - бібліотека від компанії Meta для розробки UI.

Backend - серверна частина веб-застосунку.

Frontend - клієнтська частина веб-застосунку.

JSON - (JavaScript Object Notation) - це легкий формат для обміну та зберігання даних.

Node.js - серверна платформа, обгорнута мовою JavaScript для створення маштабованих програм.

Express - каркас розробки серверної частини веб-застосунків для Node.js.

DB - “database” база даних.

ВСТУП

Що сьогодні об’єднує знімальну групу одного з найпопулярніших серіалів світу “Witcher” від Netflix та звичайну команду студентів факультету інформатики Києво-Могилянської академії. Може здатися абсурдним таке порівняння, бо на перший погляд нічого спільного між процесом зйомки серіалу та розробкою програмних продуктів не може бути. Однак, і знімальна група Netflix, і студенти КМА використовують схожі програми для керування робочими процесами командної роботи.

Сьогодні програми керування робочими процесами командної розробки стали вкрай необхідними для сучасного світу, адже з кожним днем з’являється все більше різноманітних великих проектів над якими працює велика команда, а частіше десятки невеликих команд, де кожна відповідає за певний процес. Часто саме невеликі прорахування у командній роботі і призводять до затримки випуску різноманітних проектів у реліз. Особливо це помітно у сфері розробки ігор. Я певен багато хто стикався з тим, що гру, яку він очікував понад рік переносять на кілька місяців, а то й років. Наприклад :

- Team Fortress 2 (запланований вихід: 1998, вихід: 2007)
- Mafia (запланований вихід: 2004, вихід: 2010)
- The Legend of Zelda: Breath of the Wild (запланований вихід: 2014, вихід: 2017)

Хоч сьогодні існують десятки програм для контролю командної роботи, проблема із подовженням дедлайнів досі актуальна.

Метою даної роботи є створення власної програми для контролю командної роботи, яка буде корисною для людей, що хочуть оптимізувати та контролювати процес командної розробки.

До методів дослідження відносяться :

- спостереження - пошук програм та сервісів, що дають змогу контролювати команді робочі процеси
- порівняння - знаходження плюсів кожного сервісу, порівняння функціоналу та популярності
- аналіз

РОЗДІЛ 1. АНАЛІЗ ПРОГРАМ-АНАЛОГІВ. ПОСТАНОВКА ЗАВДАННЯ КУРСОВОЇ РОБОТИ

1.1 Базовий аналіз програм-аналогів

Сьогодні ринок застосунків, що допомагають з контролем нашої щоденної активності, ефективності командної роботи розвивається дуже стрімко. Кожен з нас хоча би раз використовував такі програми, як : Trello, Asana, Slack, Jira чи Todoits. Ці мастадонти сфери тайм-трекінгу та тайм-менеджменту задають напрямок цілій індустрії програм для контролю роботи, як для однієї людини, так і для команди людей. Кожна з цих програм стала класикою у жанрі програм категорії “Продуктивності” і щоденно ними користуються мільйони людей.

Звичайно програми тайм-менеджменту мають свої особливості та підходи для планування роботи, так одні використовуються для режиму Scrum інші для режиму Agile чи Kanban.

Щоб розробити власну програму, треба проаналізувати найкращі, найпопулярніші програми, знайти їх слабкі та сильні сторони та поєднати сильні сторони у своєму застосунку.

1.1.1 Trello

Trello успадкував принципи роботи японської системи канбан. Це сервіс, що дає змогу зібрати всі завдання, ідеї та обговорення на спеціальних дошках. Ці дошки показують стан ваших задач — у них можна завантажувати файли, призначати виконавців, проводити обговорення, ставити дедлайни, додавати коментарі. На одній дошці можуть розміщуватися як маленькі, так і більші проекти з великою кількістю списків і карток.

Завдяки широкому набору інструментів можна охопити проект до деталей. Кожна окрема картка може бути як простим завданням, так і комплексним проектом, усередині якого розміщені коментарі, гіперпосилання, голосування та робочі документи.

Переваги

- Дуже легко перемістити завдання з одного списку в інший, просто перетягнувши його
- Має пряму інтеграцію з класичними інструментами версії коду, такими як github та іншими програмами
- Простий інтерфейс

Недоліки

- Функція пошуку не завжди точна для пошуку правильних дощок
- Часто запис звичайного числа трансформується у дату

1.1.2 Asana

Asana схожий за принципом сервіс до Trello . Asana створений для роботи над проектами та завданнями, для корпоративного спілкування, обміну документацією та перевірки списку справ.

Сервіс дає можливість призначати завдання, виставляти дедлайни і відстежувати статус їх виконання. Підходить для управління проектами в невеликих командах або для індивідуального використання. У межах однієї команди можна виконувати одночасно кілька проектів. Сервіс, так само, як і Trello, має канбан-дошки і може інтегруватися з іншими додатками для підвищення продуктивності.

Переваги:

- Зручний інтерфейс
- Чітко окреслює, які члени команди відповідають за кожну частину проекту у будь-який момент часу
- Наявна статистика у підписці “Premium”
- Можливість змінити вигляд проекту

Недоліки:

- Багато різноманітного функціоналу, що ускладнює використання сервісу

1.1.3 Slack

Slack — це корпоративний месенджер для листування з колегами і клієнтами. Зручно використовувати для швидкого обміну інформацією. Є можливість створювати окремі чати під кожне завдання й додавати туди тільки тих людей, яких стосується обговорення. Добре працює пошук — можна шукати як окремі повідомлення, так і чати. Сервіс підтримує інтеграцію зі сторонніми сервісами, як-от Trello, Jira, Google Drive, Dropbox.

Переваги:

- Швидкий і простий обмін повідомленнями
- Підтримка розширеного тексту для зображень, вкладень, фрагментів коду або просто виділення
- Можливість створювати десятки каналів

Недоліки

- Важко орієнтуватися, коли багато каналів

- Безкоштовний план обмежує кількість повідомлень доступних у будь-який момент часу, тобто ви можете “втратити” розмову з кимось після розмови з іншими
- Slack потребує значної кількості оперативної пам’яті

1.1.4 Jira

Сервіс для комунікації працівників, який допомагає відстежувати прорахунки. У програмі можна зазначати тематику проєкту, його класифікацію, визначати пріоритети, компоненти й контент. Для розширення задач можна використовувати додаткові поля або коментарі.

Задачі в Jira — це динамічні елементи, якими просто управляти: їх можна редагувати, змінювати статус. Також можна змінювати потік операцій, щоб визначати переходи між станами задачі. Усе, що відбувається в процесі роботи, зберігається в журналі подій.

У сервісу є також функція таймера, щоб дізнаватися, скільки часу йде на виконання задач. Дуже зручно й те, що в разі прорахунку одного з членів команди, всі учасники отримують повідомлення про помилку. Таким чином, жодна похибка не залишиться непоміченою.

Програму Jira можна інтегрувати в Telegram і отримувати повідомлення про задачі там.

Переваги:

- Наявність різноманітних схем діаграм
- Широкий спектр інтеграцій
- Можливість імпортувати звіти у Excel

Недоліки

- Зависока ціна, коли працівників більше 10 у команді
- Складні налаштування

1.1.5 Todoist

Todoist – таск-менеджер для управління завданнями проекту. Todoist дозволяє командам або окремому співробітнику структурувати робочий день або спланувати майбутні дії до списку завдань.

Командна робота в Todoist представляє собою роботу в проекті. Члени команди можуть спільно обговорювати завдання, делегувати нові завдання, встановлювати пріоритетність їх виконання, керівники проектів мають можливість встановлювати денний план співробітнику. Для перегляду вчинених дій над проектом члени команди можуть переглядати активність проекту в цілому або окремого співробітника, які завдання він виконав, а які в роботі. Команди можуть переглядати проект у вигляді дощок, що дозволяє побачити проект у розрізі і спланувати майбутні дії.

Переваги:

- Зручна інтеграція, особливо з Google Calendar
- Легко співпрацювати з товаришами по команді, членами сім’ї над різними проектами
- Можливість створити проект у двох варіаціях (Список чи Дошка)

Недоліки:

- Замало інтеграцій
- Замало безкоштовних тем та дуже обмежена кількість проектів у безкоштовному режимі

РОЗДІЛ 2. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 PERN

PERN - це стек, що складається з PostgreSQL, Express, React, Node.js. Комбінація цих технологій допомагає створити сучасний та повний стек для веб-додатку з операціями CRUD. PERN - це альтернатива MERN (MongoDB, Express, React, Node.js), що виділяється використанням реляційної бази даних PostgreSQL.

У стеці PERN React - це бібліотеека, що відповідає за веб-частину додатку, Express - це веб-фреймворк, що відповідає за серверну частину, Node.js - це середовище виконання, PostgreSQL - це реляційна база даних.

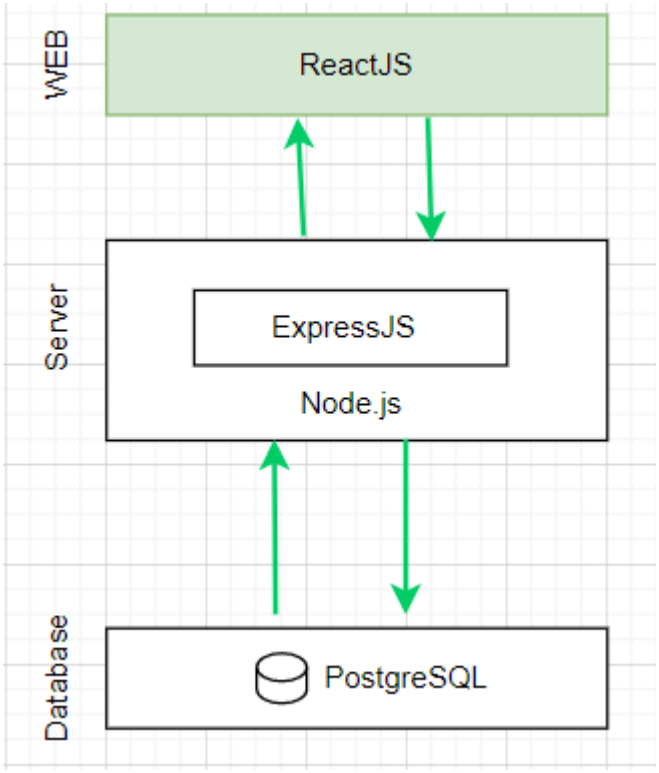

Рисунок. Трирівнева архітектура (клієнт, сервер, бд)

2.1.1 React

React - це бібліотека JavaScript, яку спеціально використовують для побудови користувальницьких інтерфейсів. Компанія Facebook запустила цю бібліотеку у 2013 році і досі ця бібліотека залишається дуже актуальною і користується колосальним попитом серед розробників. React розширює свої можливості з року в рік, тому сьогодні це повноцінна бібліотека, яка дозволяє створювати IOS, Android та веб-додатки. React пішов шляхом декларативної парадигми програмування, тобто ніколи не має взаємодії з DOM, інтерфейс користувача оновлюється, коли змінюється стан програми.

Сьогодні існує багато альтернатив для створення веб-додатків, наприклад Angular чи Vue. Та у деяких моментах React має перевагу над ними:

- **Простота**

React використовує компонентний підхід, що робить логіку розробки застосунку чіткою і простою. Компоненти чітко визначають життєвий цикл та створюють легку для коригування архітектуру. Більше того, немає розділення коду на js, html, css файли. Завдяки спеціальному розширенню JSX, можна змішувати все в одному місці.

- **Легко вчиться**

Щоб почати писати на React нам потрібні лише базові навички HTML CSS.

- **Віртуальний DOM**

Віртуальна DOM (об’єктна модель документа), яка дозволяє впорядковувати документи у форматі HTML, XHTML чи XML у дереві, що краще всього підходить для веб-браузерів для аналізу різних елементів веб-прикладів.

- **Стрімкий розвиток**

100% JavaScript бібліотека з відкритим вихідним кодом, що отримує безліч оновлень і покращень відповідно до коментарів та зауважень від розробників зі всього світу. Більше того, міграція між версіями дуже легка і не викликає значних проблем.

2.1.2 Node.js

Node.js - це серверна платформа, що зроблена на середовищі виконання JavaScript Chrome для створення масштабованих програм. Node.js написаний не на JavaScript (він написаний на C ++), але він використовує мову JavaScript як інтерпретаційну мову для обробки запиту - відповіді на стороні сервера. Node.js сьогодні використовується для швидких та масштабованих веб-додатків. Node.js сьогодні широко використовується через ряд переваг:

- **JavaScript** - Node.js дозволяє програмувати front & backend частини однією мовою, що полегшує розробку застосунку
- **Велике ком'юніті**
- **Швидкість -** Node.js асинхронна і однопоточна, тому всі операції вводу\виводу не блокують інші операції. Це означає, що можна одночасно надсилати електронні листи, читати файли та робити запити в базу даних.

Використовується для легкої та зручної побудови досить швидких та масштабованих мережевих додатків. Node.js використовує модель керовану подіями, яка не блокує введення-виведення(input-output), що робить її легкою та ефективною, гарно підходить для додатків, що працюють у режимі реального часу на розподілених пристроях.

2.1.3 Express

ExpressJS - це серверний фреймворк веб-додатків для Node.js, що призначений для створення веб-додатків та API. ExpressJS також розглядається як рамка з відкритим кодом, і вона була розроблена та підтримується фундаментом NodeJS. Сьогодні ExpressJS часто використовується як основний стек у MEAN, MERN, PERN MEVN та інших альтернативних стеках розробки повного циклу клієнт-серверних застосунків.

ExpressJS працює, використовуючи принцип middleware, тобто використовує функції, що призначені для проміжної обробки. Ці функції мають прямий доступ до об'єктів запиту(request) та відповіді(response) та до наступної функції проміжної обробки у циклі "запит-відповідь" та позначаються як **next**.

Функції проміжної роботи дозволяють вносити зміни до об'єктів запиту(request) та відповіді(response), завершувати цикл "запит-відповідь" та викликати наступного проміжного обробника зі стеку.

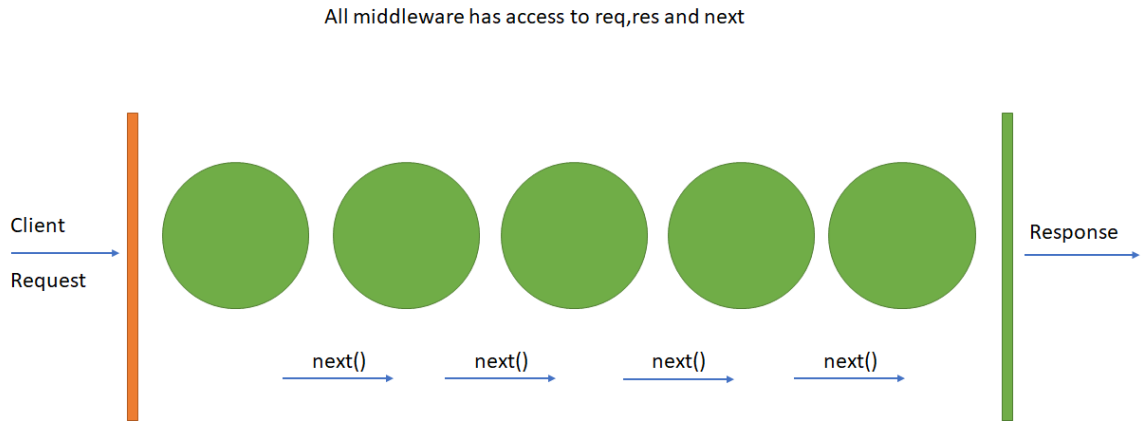

Рисунок. Процес запуску запиту до кінця функції

2.1.4 PostgreSQL

PostgreSQL - це популярна об’єктно-реляційна база даних, яка вперше вийшла ще в 1987 році. PostgreSQL не належить якійсь окремій компанії, її розробкою займається великі компанії, що зацікавлені у її використанні та інтузіасти зі всього світу. PostgreSQL є зручною базою даних, тому що підтримує паралельне виконання запитів від користувачів, використовуючи механізм MVCC (Multiversion Concurrency Control), тому виконуються принципи ACID. При використанні цієї бази даних не потрібно хвилюватися за швидкість запитів, тому що спеціальні індекси допоможуть зробити пошук більш швидшим через використання спеціальних типів: B-дерево, hash, R-дерево.

При проектуванні бази даних PostgreSQL варто дотримуватися основних правил нормалізації баз даних. Дотримання хоча би перших 3 правил нормалізації зробить зрозуміли таблиці, ефективними запити та просту архітектуру.

Хоч сьогодні спостерігається тренд на NoSQL бази даних, реляційні БД сильно вкорінилися у сучасну розробку, тому PostgreSQL і досі залишається ефективною, сучасною та надійною базою даних, що підходить як для pet-projects так і для великих комерційних рішень.

2.2 Аутентифікація через JWT

Аутентифікація - це процес встановлення того, киє являється користувач, тоді як авторизація - перевірка ролі користувача, тобто перевірка яких ресурсів користувач може мати доступ. Спочатку відбувається автентифікація, найчастіше через пошту та пароль, після чого задіюється авторизація.

Існують різні способи реалізації аутентифікації та авторизації, та найуживаніішим є спосіб використання сесій. Цей спосіб дозволяє зробити використання акаунту користувача безпечнішим, бо надає тимчасовий доступ із подальшим повтором аутентифікації та авторизації.

При роботі з сеансами активно використовують JWT(JSON WEB Token), що дозволяє створити унікальний токен при вході до застосунку.

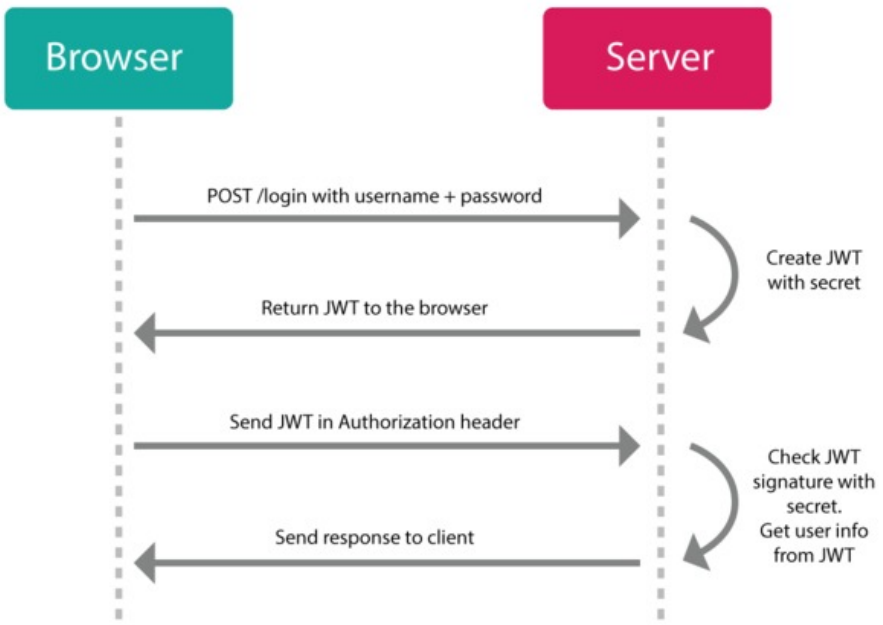

Рисунок. Авторизація через JWT токен

На рисунку продемонстровано алгоритм створення та збереження токену. Спочатку Відбувається запит на сервер, де використовуються дані із заповненої форми. Дані формують токен, що надсилається назад до браузера. Токен зберігається у файлах cookie чи у localStorage. Після успішного входу, токен буде мати вигляд Bearer <JSON WEB TOKEN>, при будь-якому запиті, що потребує авторизацію. Токен проходить перевірку, коли сервер обробляє запити до захищених ресурсів. Якщо токен варідний, то програма продовжує роботу і користувач доступ до ресурсу, інакше повертається помилка.

РОЗДІЛ 3 ОПИС ПРАКТИЧНОГО ДОСЛІДЖЕННЯ

3.1 Аналіз технічного завдання

Відповідно до завдання курсової роботи, головною метою є створення додатку, що допоміг би у керуванні робочими процесами групової розробки. Додаток має мати зручний, простий та симпатичний інтерфейс та відповідати наступним вимогам:

- Авторизація має відбуватися через JWT
- Відправка спеціального коду на пошту новому члену проекту
- Доєднання до проекту, ввівши спеціальний код, що прийшов на пошту
- Створення необмеженої кількості проектів
- Створення задачі та призначення відповідального
- Встановлення таймеру виконання задачі
- Перегляд та сортування всіх завдань

3.2 Проектування моделі даних

Проектування та реалізація бази даних PostgreSQL відбувалася у спеціальному застосунку DataGrip. Сама база даних автоматично генерується завдяки бібліотеці sequalize, де ми проектуємо об’єкти у класі models.js, вказуючи параметри та властивості таблиць.

```
const sequelize = require('../db')
const {DataTypes} = require('sequelize')

const Person = sequelize.define('people', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
    name: {type: DataTypes.STRING},
    email: {type: DataTypes.STRING},
    password: {type: DataTypes.STRING}
})
```

Рисунок. приклад моделі Person

За допомогою інструментів DataGrip було відображено схему бази даних.

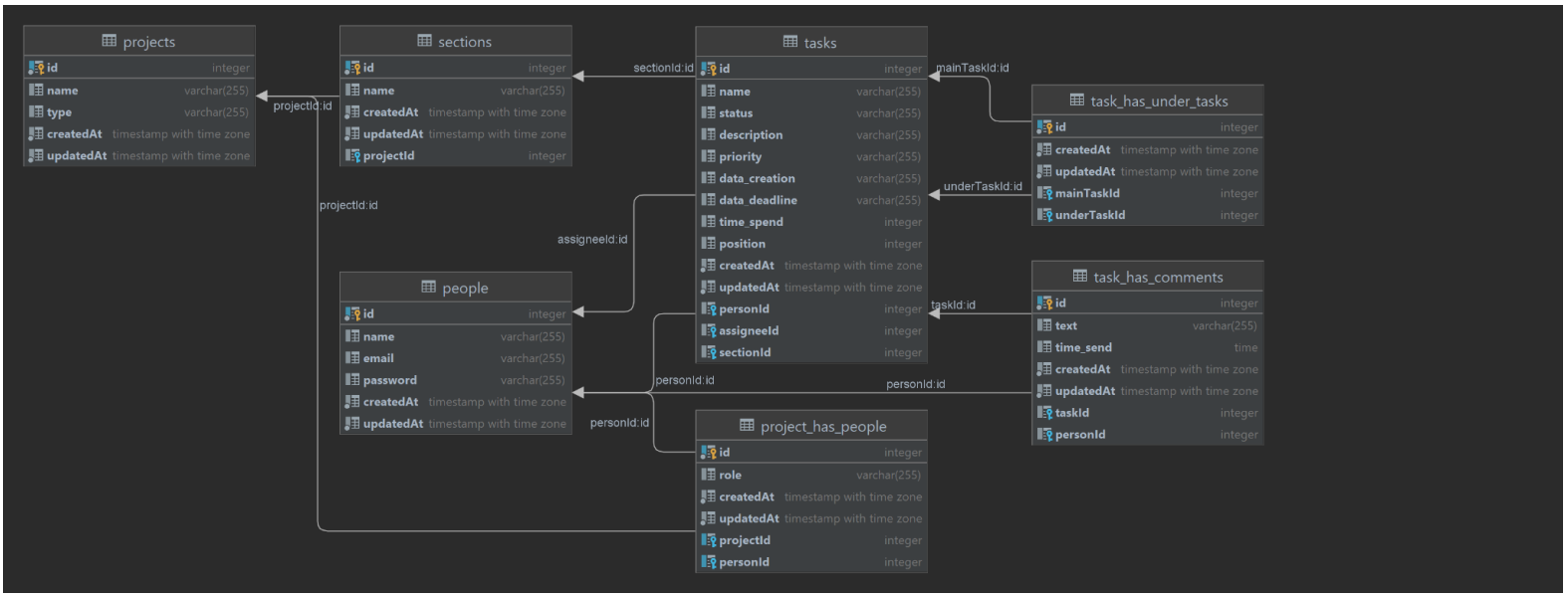

Рисунок. Схема бази даних

3.3 Реалізація серверної частини

Цей розділ присвячений серверній частині застосунку, що написаний на Node.js Express.

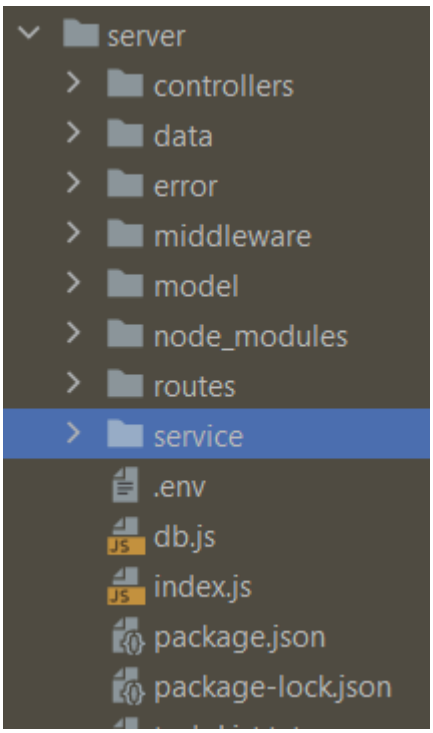

Рисунок. Структура проекту серверної частини

Рисунок зображує лише головні папки серверної частини проекту, бо підпапки проекту не влізли до загального вікна. Структуру проекту умовно можна поділити на такі частини:

- класи контролери(controllers)
- клас дефолтних даних (data)
- класи помилок (error)
- рівень middleware (middleware)
- клас моделей бази даних (model)
- класи роутів (routes)
- класи сервісів (services)

3.3.1 Контролери та сервіси

Основна бізнес логіка знаходиться саме у класах контролерах та сервісах. Застосунок побудований на базі 3-рівневої архітектури, де на

найвищому рівні знаходяться контролери, які відповідають за взаємодію між url та методами запитів. Класи контролери мають головний файл index.js із папки (routes), що відповідає за просту маршрутизацію між контролерами.

```
const Router = require('express')
const router = new Router()
const projectRouter = require('./projectRouter')
const homeRouter = require('./homeRouter')
const myTasksRouter = require('./myTasksRouter')
const userRouter = require('./userRouter')

router.use('/projects', projectRouter);
router.use('/home', homeRouter);
router.use('/mytasks', myTasksRouter);
router.use('/user', userRouter)
module.exports = router
```

Рисунок. головного класу index.js(routes), де йде розподіл

Класи сервіси відповідають за бізнес логіку застосунку та допомагають прибрати зайву логіку із контролером. Сервіси виступають проміжним логічним шаром між контролерами та доступом до баз даних.

```
export default class UserService() {

    async Signup(user) {
        const userRecord = await this.userModel.create(user);
        const companyRecord = await this.companyModel.create(user);
        this.eventEmitter.emit('user_signup', { user: userRecord, company: companyRecord }
        return userRecord
    }
}
```

Рисунок. ласу сервісу UserService

3.3.2 Помилка

Класи помилки використовуються для власного локального виявлення проблем. Є кілька підходів у створенні класів помилок:

- створення класу під кожну помилку
- створення одного класу із параметром message, для опису суті проблеми

У курсовій роботі використовується другий варіант із створенням одного класу ApiError, що використовуються у класах middleware.

3.3.3 Middleware

Класи middleware, використовуються разом з контролерами у ролі фільтрів, щоб перевірити вхідні параметри і виконати метод, якщо параметри відповідають вимогам. У застосунку створено кілька middleware рівнів, для перевірки та виявлення помилок при доступу до недозволених ресурсів.

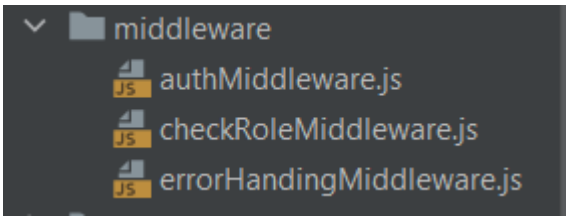

Рисунок. Класи middleware

```
router.post( path: '/section', projectController.createSection)
router.post( path: '/task', projectController.createTask)
router.post( path: '/:projectId', checkRole( role: "ADMIN"), projectController.create)
```

Рисунок. застосування middleware у класів контролеру

3.3.4 Моделі

У проекті наявний файл model.js, що завдяки бібліотеці sequalize створює моделі, які описують таблиці бази даних. У даному файлі знаходяться всі моделі, вони не розділені на окремі файли, як це практикується у Java Spring Boot.

```
const Person = sequelize.define('people', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},
    name: {type: DataTypes.STRING},
    email: {type: DataTypes.STRING},
    password: {type: DataTypes.STRING}
})
```

Рисунок.Приклад генерації таблиці people

Як видно зі скриншоту, тут описується створення таблиці people, що має поля id, name, email, password, де id - це первинний ключ таблиці.

3.3.5 Router

Маршрутизація відбувається у папці routes. Головний файл index.js, відповідає за маршрутизацію між логічно розділеними роутерами.

```
router.use('/projects', projectRouter);
router.use('/home', homeRouter);
router.use('/mytasks', myTasksRouter);
router.use('/user', userRouter)
module.exports = router
```

Рисунок. Маршрутизація веб-додатку

Сам клас роутерів відповідає за методи запитів http, коректні url, рівень middleware та запитами із контролерів

```
const Router = require('express');
const router = new Router();
const projectController = require("../controllers/projectController");
const checkRole = require("../middleware/checkRoleMiddleware")


router.get('/code', projectController.getProjectCode)
router.get('/:projectId', projectController.getProjectInfo);

router.post( path: '/section', projectController.createSection)
router.post( path: '/task', projectController.createTask)
router.post( path: '/:projectId', checkRole( role: "GUEST"), projectController.create)

router.delete( path: '/section', projectController.deleteSectionById)
router.delete( path: '/task', projectController.deleteTaskById)

router.put( path: '/section', projectController.updateSection)
router.put( path: '/task', projectController.updateTask)
```

Рисунок. Запити у класі ProjectRouter.js

Роутери підтримують всі методи запитів: get, post, put, head, delete, options, trace, copy, propfind, proppatch, unlock, repost, mkactivity, lock, mkcol, move, purge, checkout, notify, subscribe, merge, m-search, unsubscribe, patch, search, connect. У курсовій роботі використовувалися лише основні методи: get, post, put, delete.

3.4 Реалізація клієнтської частини

Розглянемо клієнтську частину, що реалізована на React. Як зазначалося вище, React - це javascript бібліотека, що використовується для побудови клієнтських інтерфейсів. Застосунок на React будується через структуру компонентів.

```
const Task = (props) => {
    const [task, setTask] = useState(props.task)
    const [show, setShow] = useState( initialState: false)
    return (
            <tr>
                <td onClick={() => setShow( value: true)}>
                    {task.name}
                    <TaskModalEdit
                        deleteTask={props.deleteTask}
                        updateTask={props.updateTask}
                        onClose={() => setShow( value: false)}
                        show={show}
                        task={props.task}
                    />
                </td>
                <td>{task.assigneeId}</td>
                <td>{task.data_deadline}</td>
                <td>{task.priority}</td>
                <td>{task.status}</td>
            </tr>

    );
};
```

Рисунок. приклад функціонального компоненту React

Всі компоненти у застосунку функціональні, через свою зручність та можливість використовувати react hooks. Головним стартуючим компонентом є клас index.js, що знаходиться у кореневій папці src.

```
export const Context = createContext( defaultValue: null);

ReactDOM.render(
    <React.StrictMode>
        <Context.Provider value={{
            user: new UserStore(),
            projectStore: new ProjectStore()
        }}>
            <App/>
        </Context.Provider>
    </React.StrictMode>,
    document.getElementById( elementId: 'root')
);
```

Рисунок. стартуючий файл react застосунку

3.4.1 Перелік використаних бібліотек

Всі бібліотеки допоміжні встановлювалися за допомогою менеджеру пакетів npm і зберігалися у папці devDependencies. Найуживаніші бібліотеки:

- @material-ui - бібліотека із симпатичними HTML елементами
- axios - бібліотека для виконання HTTP-запитів
- jwt-decode - бібліотека для декодування токену
- mobx-react - бібліотека допомагає реагувати на будь-які зміни стану
- react-icons - бібліотека включає популярні значки та іконки, які леко використовувати у проекті

3.4.2 Архітектура застосунку

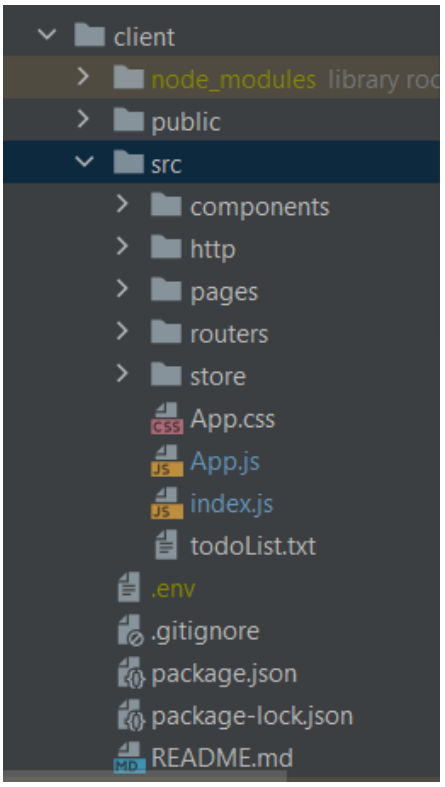

Рисунок. Схема react додатку

React додаток має стандартну структуру:

- components - папка, що зберігає компоненти. Наприклад компонент Navbar
- http - папка, що відповідає за HTML-запити
- pages - папка, що зберігає функціональні компоненти сторінок
- routes - папка, що відповідає за розподіл url між ролями користувачів
- store - папка, що містить базові структури (userStore, ProjectStore)

3.5 Тестування програми та результати її виконання

Перед початком тестування застосунку, я створив автоматичну генерацію фейкових даних, які щоразу при запуску сервера заново автозаповнюють базу даних. Фейкові дані створені у невеликій кількості, але їх достатньо, щоб протестувати та перевірити весь функціонал застосунку. Кожен користувач має унікальну пошту і однаковий захешований пароль.

Коли користувач входить на сайт, він бачить головну сторінку, де зображено актуальну дату, побажання приємного дня та назву застосунку.

Рисунок. Сторінка "Home"

При натиску на кнопку "Login" користувача переносить на сторінку аутентифікації, де наявні форма вводу пошти та паролю, кнопка "Login" та кнопка "Registration", якщо користувач хоче створити акаунт.

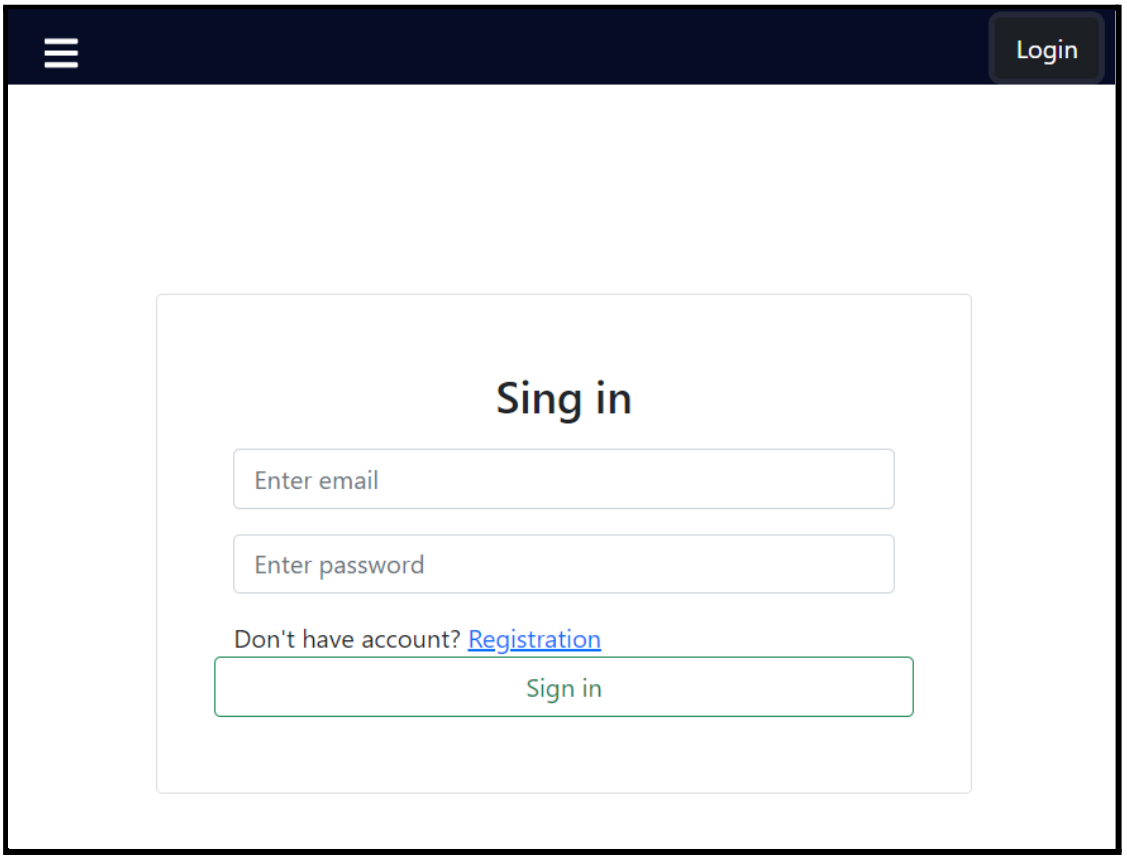

Рисунок. Сторінка "Login"

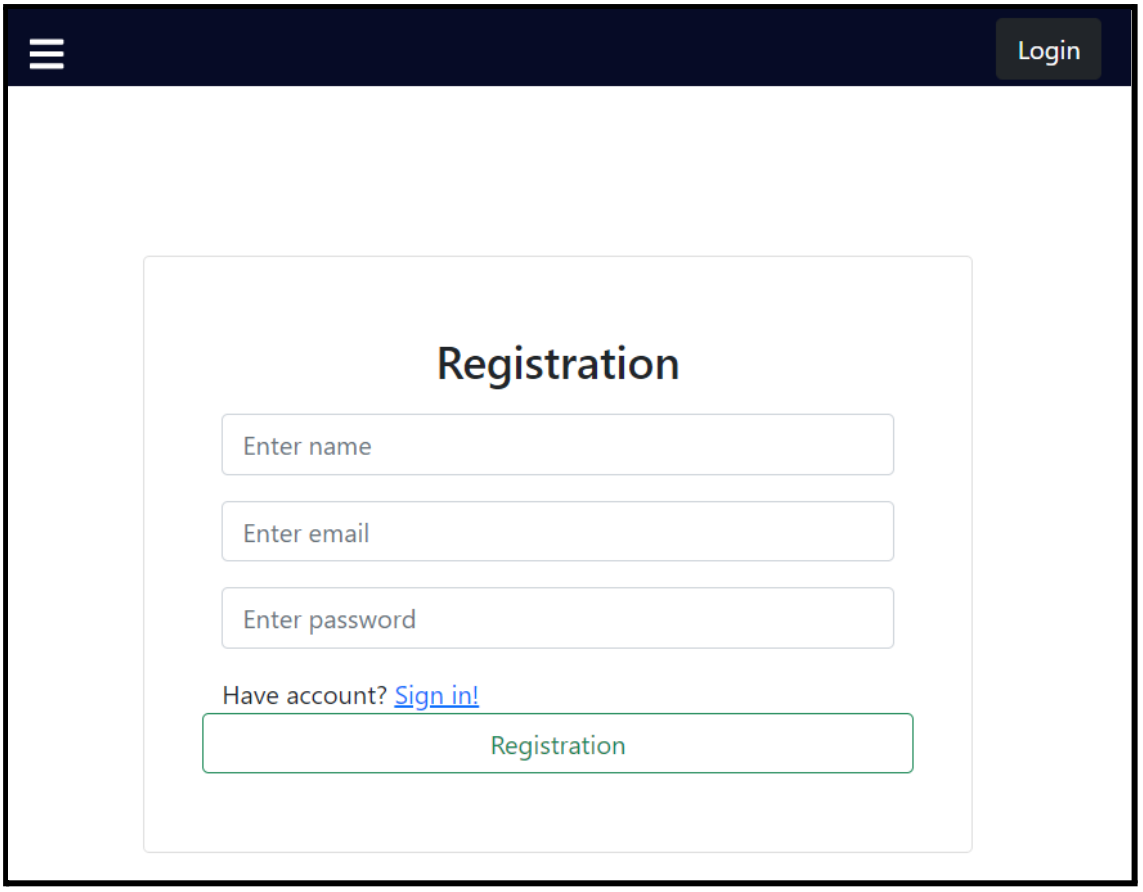

Рисунок. Сторінка "Registration"

Пройшовши аутентифікацію та авторизацію за поштою та паролем, користувач потрапляє на сторінку своїх задач.

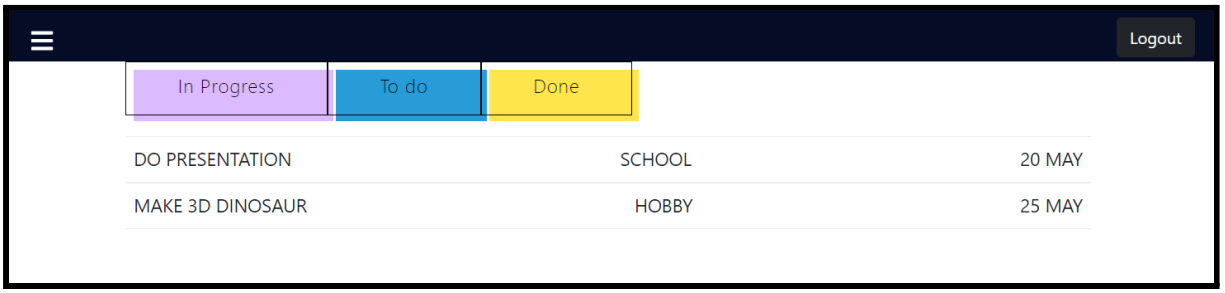

Рисунок. Сторінка "Tasks"

На сторінці "Tasks" наявний список із всіх задач користувача, що поділені за статусами. Користувач може переглянути свої задачі, натиснувши на одну із кнопок наявних статусів.

Зверху розташований спеціальний компонент “navbar” із спеціальним випадаючим меню, що допомагає пересуватися сайтом.

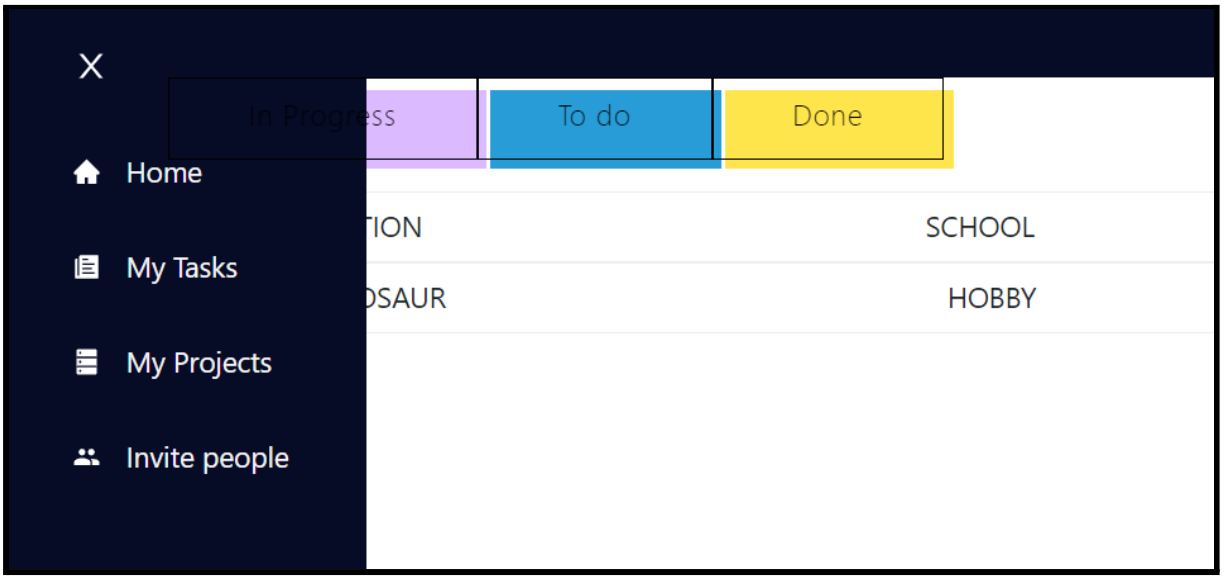

Рисунок. Випадаюче меню

У меню у нас наявні наступні переходи:

- Home (Головна сторінка)
- My Tasks (Всі проекти)
- My Projects (Створення проекту)
- Invite people (Запросити людину)

Коли користувач переходить на сторінку “My Projects”, то виводиться список із всіма проектами користувача. Щоб перейти на певний проект, потрібно натиснути на назву проекту.

Рисунок. Сторінка “My Projects”

Натиснувши на назву проекту, користувач переходить на сторінку обраного проекту.

Рисунок. Сторінка "Project"

На сторінці проекту, можна створити певну секцію та додати до цієї секції задачі.

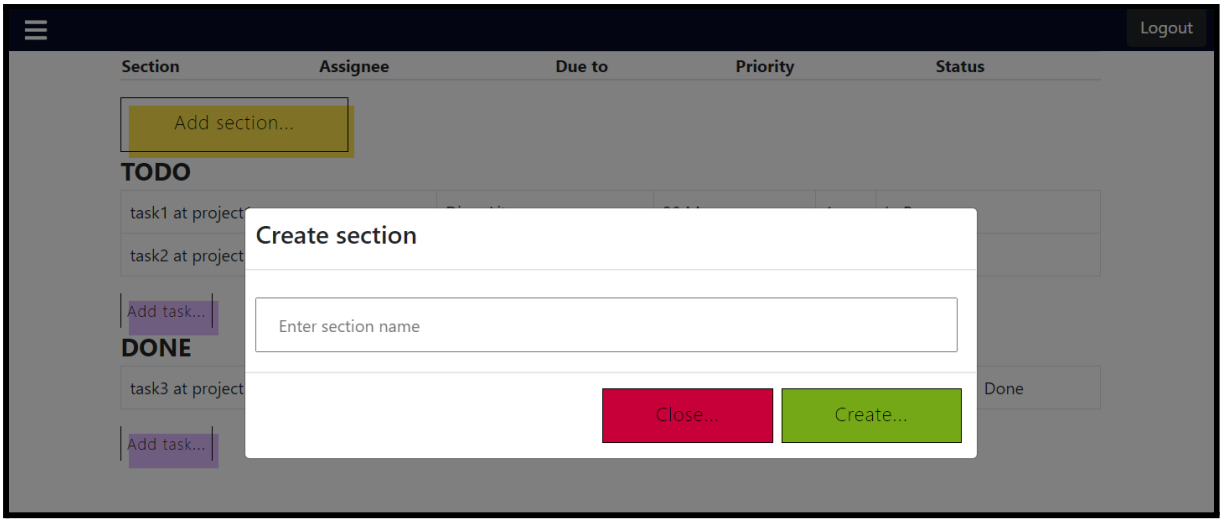

Рисунок. Модульне вікно створення секції

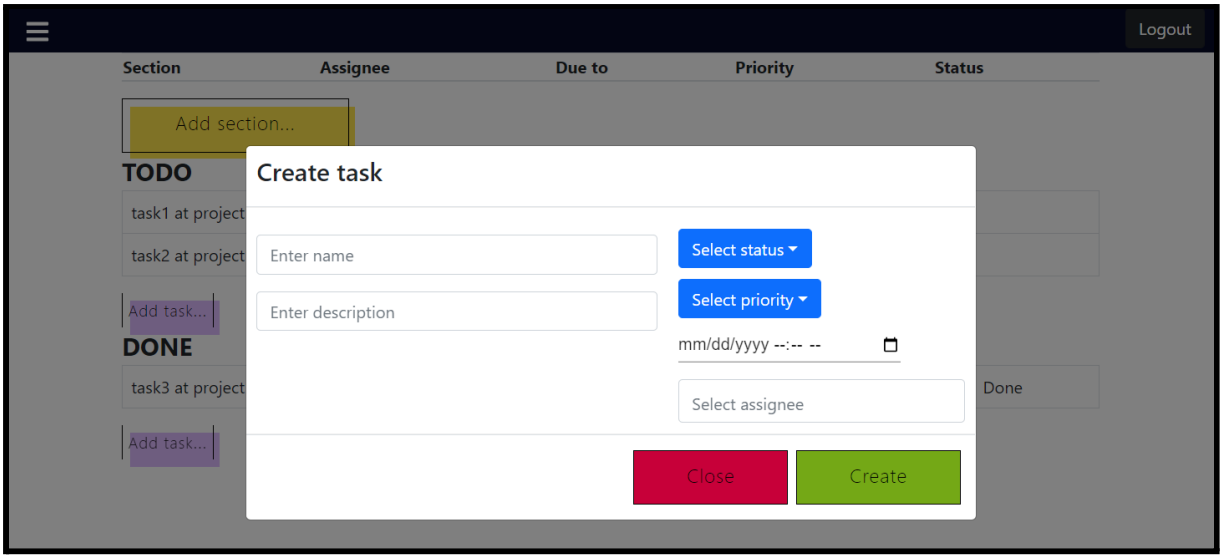

Рисунок. Модульне вікно для створення таски

При натиску на назву секції чи таски, з'являється спеціальне модальне вікно, що дозволяє змінити дані:

- секція - змінити назву
- таска - змінити назву, дедлайн, статус, пріоритет, виконувача, видалити таску, перейти на сторінку таймеру цієї задачі

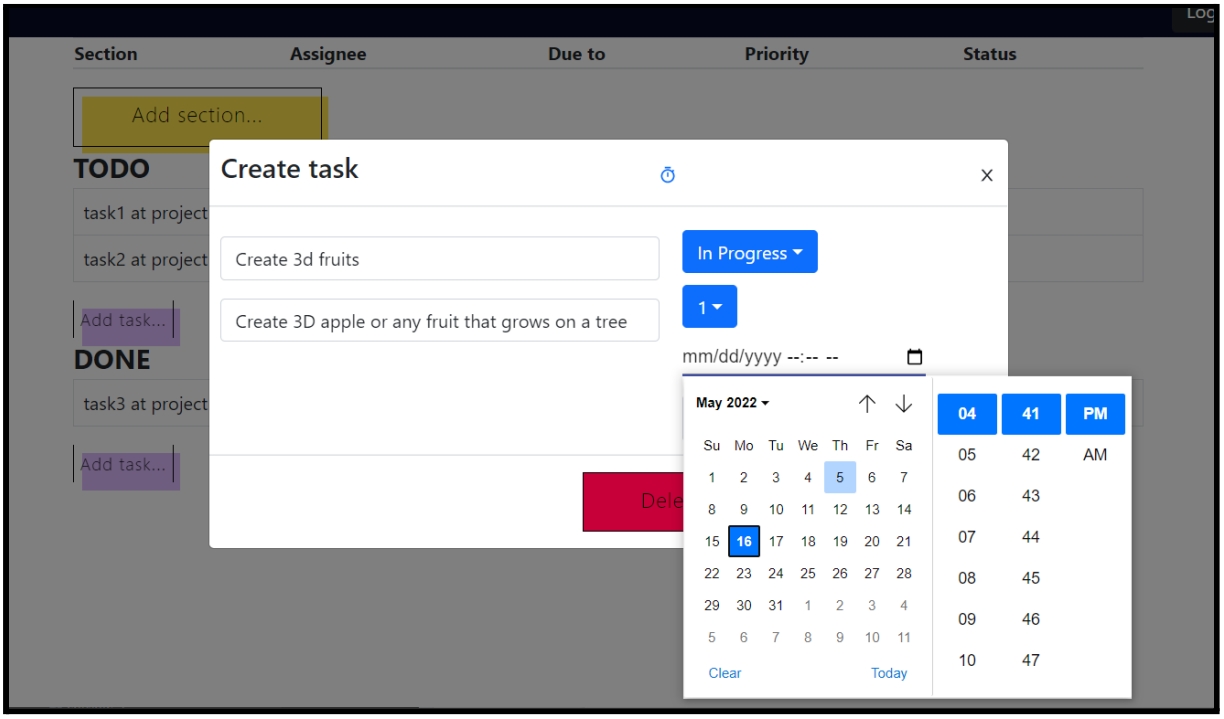

Рисунок. Модульне вікно для редагування параметрів таски

У модальному вікні таски наявна кнопка таймеру, що перенаправляє користувача на спеціальну сторінку-таймер, яка відповідає за рахунок часу, витраченого на виконання певної задачі. Сторінка-таймер дозволяє запускати таймер та переглядати статистику витраченого часу на задачу.

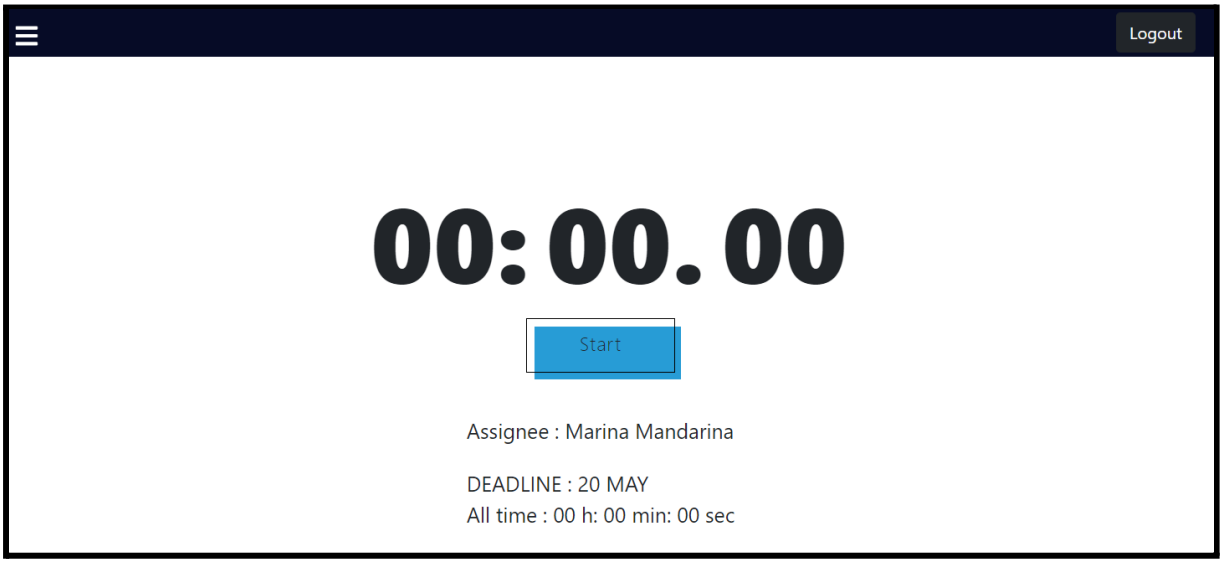

Рисунок. Сторінка "Task-timer"

Остання доступна сторінка - це сторінка "Invitation". На цій сторінці знаходяться дві кнопки. перша відкриває форму для запрошення людини на проект, за поштою користувача. Тобто потрібно вибрати проект із списку та ввести пошту користувача, якого планують запросити до команди. Цій людині на пошту прийде повідомлення із спеціальним кодом на приєднання до проекту. Друга кнопка відкриває форму власне для приєднання до проекту. У цій формі потрібно ввести код, що прийшов вам на пошту та натиснути кнопку "Enter project".

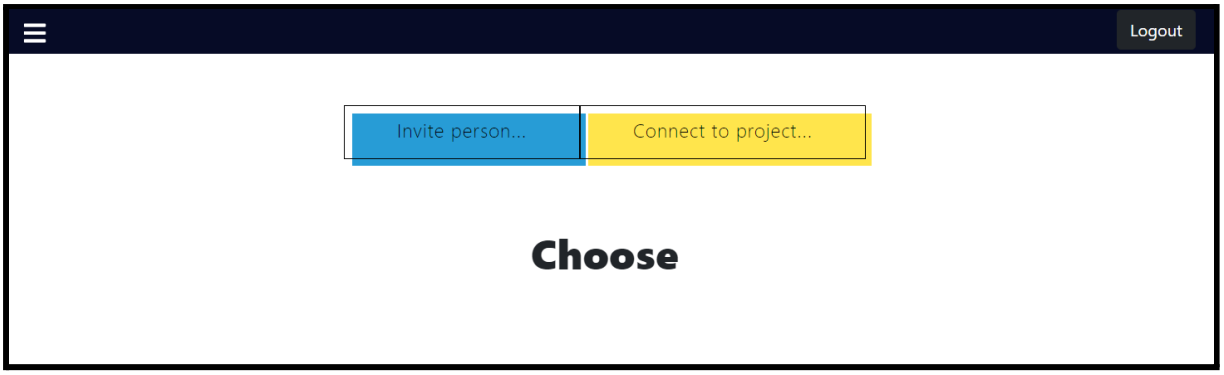

Рисунок. Сторінка "Invitate people"

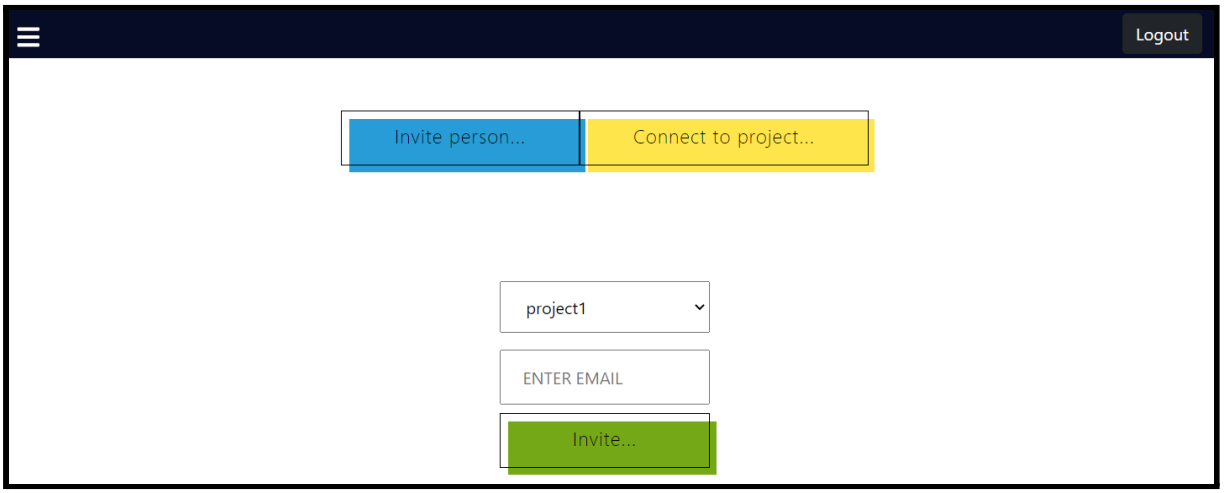

Рисунок. Сторінка "Invite people" форма запрошення до проекту

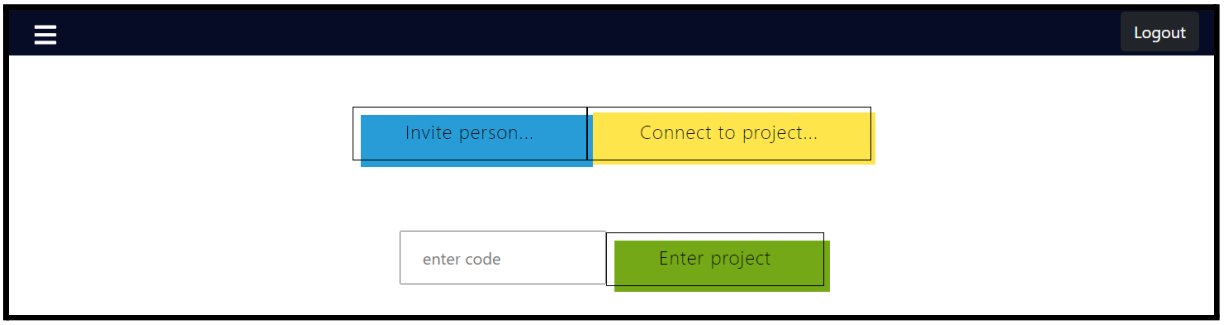

Рисунок. Сторінка "Invite people" форма приєднання до проекту

Висновки

У процесі розробки курсової роботи мною було проаналізовано процес створення застосування для організації розробки та керування робочими процесами.

Розробляючи програму для організації командної роботи, було досліджено та проаналізовано програм-аналогів, досліджено як сильних сторін програм, так і їх мінусів.

Найдоцільнішим варіантом для реалізації даного застосунку було створення веб-сайту. Тож було реалізовано веб-сайт, який дозволяє створити та упорядочити задачі на проектах. Унікальною складовою застосунку є спеціальна сторінка “Таймер задачі”, яка дозволяє відслідковувати час, затрачений на виконання певної задачі. Веб-додаток має зручний та інтуїтивно зрозумілий дизайн, який освоїть навіть не обізнаний користувач.

Проект має багато шансів для подальшого розвитку та вдосконалення. БД проектувалася за правилами нормалізації, щоб легко та невитратно її розширювати.

Цей веб-додаток є альтернативою додаткам для організації та керуванням робочими процесами.

Посилання

1. Застосунок Asana. [Електронний ресурс]. Режим доступу: https://app.asana.com/
2. Застосунок Trello. [Електронний ресурс]. Режим доступу: https://trello.com/ru
3. Застосунок Slack. [Електронний ресурс]. Режим доступу: https://slack.com/
4. Застосунок Jira. [Електронний ресурс]. Режим доступу: https://www.atlassian.com/ru/software/jira
5. Застосунок Todoits. [Електронний ресурс]. Режим доступу: https://todoist.com/app/today
6. Створення моделі об’єкту [Електронний ресурс]. Режим доступу: https://sequelize.org/docs/v6/core-concepts/model-basics/
7. Маршрутизація Express [Електронний ресурс]. Режим доступу: https://expressjs.com/ru/guide/routing.html
8. Контролери та маршрутизація Express [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/ru/docs/Learn/Server-side/Express_Nodejs/routes
9. PERN stack [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/what-is-pern-stack/
10. JWT code\decode [Електронний ресурс]. Режим доступу: https://jwt.io/
11. How to use JSON WEB TOKEN (JWT) in Express [Електронний ресурс]. Режим доступу: https://www.digitalocean.com/community/tutorials/nodejs-jwt-expressjs
12. Sequalize create query [Електронний ресурс]. Режим доступу: https://sequelize.org/docs/v6/core-concepts/raw-queries/

13. Map in React [Електронний ресурс]. Режим доступу: https://ru.reactjs.org/docs/lists-and-keys.html
14. Lucid for database [Електронний ресурс]. Режим доступу: https://lucid.app/documents#/dashboard
15. React hooks [Електронний ресурс]. Режим доступу: https://ru.reactjs.org/docs/hooks-intro.html
16. React axios [Електронний ресурс]. Режим доступу: https://tech-wiki.online/ru/axios.html#:~:text=%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B2%20Axios-,Axios%20%2D%20%D0%BE%D1%87%D0%B5%D0%BD%D1%8C%20%D0%BF%D0%BE%D0%BF%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%B0%D1%8F%20%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0%20JavaScript%2C%20%D0%BA%D0%BE%D1%82%D0%BE%D1%80%D1%83%D1%8E%20%D0%B2%D1%8B%20%D0%BC%D0%BE%D0%B6%D0%B5%D1%82%D0%B5%20%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D1%8C%20%D0%B4%D0%BB%D1%8F,%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B0%D1%8F%20%D0%BF%D0%BE%D0%B4%D0%B4%D0%B5%D1%80%D0%B6%D0%BA%D1%83%20IE8%20%D0%B8%20%D0%B2%D1%8B%D1%88%D0%B5.
17. React material-ui [Електронний ресурс]. Режим доступу: https://mui.com/material-ui/getting-started/installation/
18. React icons [Електронний ресурс]. Режим доступу: https://react-icons.github.io/react-icons/
19. Використання observer [Електронний ресурс]. Режим доступу: https://habr.com/ru/post/348960/
20. UseEffect React [Електронний ресурс]. Режим доступу: https://ru.reactjs.org/docs/hooks-effect.html

21.4 Ways t add Styles to React Component [Електронний ресурс]. Режим доступу: https://medium.com/technofunnel/4-ways-to-add-styles-to-react-component-37c2a2034e3e