Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут деревообробних та комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету(відділення))

Кафедра інформаційних систем та комп'ютерного моделювання
(повна назва кафедри (предметної циклової комісії))

Пояснювальна записка

до дипломної роботи
перший (бакалаврський)
(рівень вищої освіти)

на тему "**Розроблення вебзастосунку для організації особистого часу (дозвілля) з використання js (Frontend)**"

Виконав: студент IV курсу, групи ІСТ- 41
спеціальності
126 "Інформаційні системи і технології"
(шифр і назва напряму підготовки, спеціальності)
Святобог Г.О.
(прізвище та ініціали)
Керівник **доц. Мокрицька О.В.**
(прізвище та ініціали)
Рецензент Думанський О.Ї.
(прізвище та ініціали)

Львів – 2023

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ деревообробних та комп'ютерних технологій і дизайну
Кафедра інформаційних систем та комп'ютерного моделювання
Рівень вищої освіти перший (бакалаврський)
Спеціальність 126 " Інформаційні системи та технології"
(шифр і назва)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІСКМ
Сторожук О.Л.
"21" 11 2022 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Святобог Галині Олександрівні
(прізвище, ім'я, по батькові)

1. Тема роботи *«Розроблення вебзастосунку для організації особистого часу (дозвілля) з використання js (Frontend)»*
керівник роботи *Мокрицька Ольга Володимирівна, к.т.н., доцент*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від *"21" 11 2022 року № С-521*

2. Термін подання студентом роботи *10.06.2023р.*

3. Вихідні дані до роботи: *Формулювання задачі та її формалізація. Аналіз попередніх досліджень. Огляд програмних засобів для реалізації поставленого завдання.*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):

1) Стан проблемної області;

2) Інформаційне та математичне забезпечення;

3) Програмне та технічне забезпечення.

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

Слайди доповіді, актуальність теми, постановка завдання, аналіз отриманних результатів, висновки

6. Дата видачі завдання *23 листопада 2022 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примі
1	Огляд літературних та інших джерел згідно досліджуваної теми. Збір потрібних матеріалів.	24.11.2022-30.11.2022 р.	Виконано
2	Постановка задачі та її формалізація.	01.12.2022-15.12.2022 р.	Виконано
3	Вибір та обгрунтування методів і засобів розв'язання завдання.	16.12.2022-16.01.2023 р	Виконано
4	Програмна реалізація системи.	17.01.2023-17.04.2023 р.	Виконано
5	Оформлення опису створеної програми.	18.04.2023-18.05.2023 р.	Виконано
6	Аналіз отриманих результатів виконання програми.	19.05.2023 р.	Виконано
7	Здача пояснювальної записки на перевірку та виправлення виявлених помилок.	20.05.2023-10.06.2023р.	Виконано

Студент _____ Святобог Г.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Мокрицька О. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 40 сторінок пояснювальної записки, 7 рисунків та 15 джерел.

Дипломна робота на тему "Розробка веб-застосунку для організації власного часу з використанням мови програмування JavaScript та базою даних" є високоактуальною та важливою у зв'язку зі зростанням потреб у людей у зручному та ефективному плануванні та організації свого часу в умовах швидкоплинного життя.

Метою дипломної роботи є розробка веб-застосунку, який дозволяє користувачам організовувати свій час та планувати свої задачі та події. Веб-застосунок повинен мати інтерфейс користувача, який дозволяє користувачам створювати, редагувати та видаляти свої завдання та події, а також відслідковувати їх стан та категорії.

Для досягнення поставленої мети було використано мову програмування JavaScript та базу даних. У роботі детально розглянуто процес розробки веб-застосунку, включаючи проектування архітектури, створення інтерфейсу користувача, налаштування серверу та бази даних.

Веб-застосунок розроблено з використанням Node.js та Express.js для розробки серверної частини та MongoDB як бази даних. У дипломній роботі детально описано структуру бази даних, яка містить інформацію про користувачів, завдання, події, категорії, пріоритети та статистику виконання.

Ключові слова*:* JavaScript, Node.js, Express.js, Mongo.DB, веб застосунок.

ABSTRACT

The thesis contains 40 pages of an explanatory note, 7 figures and 15 sources.

The diploma thesis on "Development of a web application for personal time organization using JavaScript programming language and a database" is highly relevant and important due to the growing need for people to have a convenient and effective planning and organization of their time in the fast-paced world.

The aim of the diploma thesis is to develop a web application that allows users to organize their time and plan their tasks and events. The web application should have a user interface that allows users to create, edit, and delete their tasks and events, as well as track their status and categories.

To achieve the set goal, JavaScript programming language and a database were used. The paper thoroughly examines the process of developing a web application, including architecture design, user interface creation, server and database configuration.

The web application was developed using Node.js and Express.js for server-side development and MongoDB as a database. The diploma thesis provides a detailed description of the database structure, which contains information about users, tasks, events, categories, priorities, and performance statistics.

Keywords: JavaScript, Node.js, Express.js, Mongo.DB, web application.

ТЕХНІЧНЕ ЗАВДАННЯ

Технічне завдання на розробку веб-застосунку для організації власного часу з використанням мови програмування JavaScript та базою даних.

Метою проекту є створення веб-застосунку, який дозволить користувачам організовувати свій час та планувати свої завдання та події. Функціонал включає можливість створення, редагування та видалення завдань та подій, а також відслідковування їх стану та категорій. Для реалізації проекту використовується мова програмування JavaScript та база даних. Розробляється інтерфейс користувача, серверна частина застосунку з використанням Node.js та Express.js, а також база даних на основі MongoDB. Проект передбачає створення структури бази даних, яка міститиме інформацію про користувачів, завдання, події, категорії, пріоритети та статистику виконання. Ключові слова: JavaScript, Node.js, Express.js, MongoDB, веб-застосунок.

ЗМІСТ

ВСТУП

В сучасному світі люди постійно поспішають, намагаючись встигнути виконати все, що потрібно, проте не завжди успішно організовують свій час. Відсутність планування та недостатня систематизація завдань часто призводять до стресу, прогалин у роботі та втрати часу на непотрібні справи. Саме тому, розробка веб-застосунку для організації власного часу є актуальною та перспективною темою дослідження.

У контексті сучасних вимог до продуктивності та ефективності, використання технологій, таких як JavaScript та бази даних, стає необхідністю для створення веб-застосунку, який допомагатиме людям в організації їхнього часу. Такий веб-застосунок дозволить користувачам легко планувати свої завдання, визначати важливі дати та події, а також зберігати і контролювати свої розклади. Він стане незамінним помічником у підвищенні продуктивності та досягненні поставлених цілей.

Метою даної роботи є створення функціонального веб-застосунку для планування та систематизації завдань, який дозволить користувачам ефективніше використовувати свій час та зменшити кількість стресових ситуацій, пов'язаних з організацією робочих та особистих справ. Для досягнення цієї мети, у роботі будуть розглянуті такі завдання: аналіз потреб і вимог користувачів, розробка функціональних вимог до системи, вибір та використання відповідних технологій, розробка імплементація системи, тестування та документування.

Предметом дослідження є аналіз сучасних підходів до розробки веб-застосунків та їх використання для розробки систем організації часу, а також проектування та розробка функціональної системи, що задовольняє потреби користувачів у плануванні та систематизації завдань. Дослідження включатиме аналіз існуючих веб-застосунків, їх функціональності та можливостей, а також вибір та застосування найкращих практик у розробці.

Практичне значення дослідження полягатиме у можливості використання розробленого веб-застосунку для покращення організації власного часу користувачами. Він надасть їм можливість зручного створення розкладів, визначення важливих дат та подій, а також ефективного контролю над завданнями. Крім того, розроблений застосунок може слугувати основою для подальшого розширення та удосконалення функціоналу, зокрема для інтеграції з іншими застосунками чи сервісами, що робить його більш універсальним та цінним для різних груп користувачів.

Отже, дана робота визначається актуальністю проблеми організації власного часу, що вимагає розробки веб-застосунку для ефективного планування та систематизації завдань. Цей дипломний проект спрямований на розробку функціональної системи, яка забезпечить користувачам зручність, ефективність та контроль над їхнім часом. В результаті дослідження та розробки очікується створення інструменту, що знадобиться для покращення продуктивності та досягнення поставлених цілей.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд проблемної області.

Сучасний ритм життя вимагає від людей ефективно організовувати свій час. Виникає потреба у зручному та доступному засобі для планування задач та контролю за їх виконанням. З цією метою розробляються різноманітні додатки та веб-застосунки.

Однією з найпоширеніших проблем є поганий менеджмент часу та відсутність системи планування робочого дня, що призводить до стресу, втоми та низької продуктивності. Відсутність чіткого розподілу часу може стати перешкодою на шляху до досягнення поставлених цілей та завдань.

Для вирішення цієї проблеми розробляються різноманітні додатки та веб-застосунки для організації власного часу. Застосунки такого типу дозволяють користувачам створювати список завдань, планувати свій робочий день та контролювати виконання задач.

Огляд існуючих рішень показує, що на сьогоднішній день існує значна кількість додатків та веб-застосунків для організації власного часу, але більшість з них мають певні обмеження або не відповідають усім вимогам користувачів.

Таким чином, необхідно розробити новий веб-застосунок для організації власного часу, який буде забезпечувати максимальний функціонал та задовольняти вимоги користувачів.

1.2. Актуальність розроблення застосунку

Розробка веб-застосунку для організації власного часу є актуальною задачею в сучасному світі, де швидкий ритм життя і велика кількість різних завдань, що потребують виконання, створюють необхідність в ефективному плануванні та управлінні часом.

Незважаючи на наявність різноманітних інструментів та додатків для організації часу, їхні можливості часто обмежені або не задовольняють потреб

користувачів. Також, багато з існуючих рішень вимагають сплати абонентської плати або не дозволяють повністю налаштувати індивідуальні потреби користувача.

Існують декілька веб-застосунків та мобільних додатків, призначених для допомоги в організації часу та плануванні завдань. Деякі з них є загальновідомими, наприклад, Google Calendar, Todoist, Trello, Asana, Microsoft To Do та багато інших.

Проте, деякі з них мають обмеження у використанні, наприклад, потребують платні плани або обмеження у функціоналі без підписки. Деякі додатки можуть бути складні у використанні, занадто спрощеними або не мають певних функцій, які користувачі хотіли би мати.

Саме тому потреба у розробленні нового веб-застосунку зростає і набуває актуальності, який буде спрощеним у використанні та матиме всі необхідні функції для планування завдань та організації часу. Розробка такого застосунку дозволить користувачам легко організувати свій час, планувати завдання, контролювати прогрес та досягати поставлених цілей.

Отже, розробка веб-застосунку, який дозволить ефективно організовувати час і враховувати індивідуальні потреби користувачів, є актуальною задачею в сфері інформаційних технологій. Такий застосунок може допомогти користувачам зосередитись на пріоритетних завданнях, запобігти зайвим витратам часу та підвищити продуктивність. Він також може бути корисним для студентів, які хочуть ефективно керувати своїм навчанням, працівників, які мають багато завдань на роботі, або будь-якої іншої людини, яка хоче ефективно управляти своїм часом.

Крім того, використання веб-застосунків також може призвести до зменшення кількості використаної паперової документації. Це може допомогти зменшити витрати на друк та зберігання паперових документів, а також зменшити кількість відходів, що утворюються при виготовленні та переробці паперу.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Дослідження цільової аудиторії

Цільова аудиторія - це базове поняття, яке використовується в маркетингу, для позначення певної кількості людей, які об’єднані спільними інтересами, потребами або темами. Цільова аудиторія бренду - це ті люди, які можуть стати потенційними покупцями того чи іншого товару/послуги [1].

Перш за все дослідження цільової аудиторії є важливою та невід'ємною складовою процесу створення веб-застосунку для організації власного часу з використанням мови програмування JavaScript та бази даних. Воно має на меті зрозуміти потреби та вимоги користувачів, спрямувати розробку в правильному напрямку та забезпечити успішне впровадження продукту. Дослідження цільової аудиторії допомагає визначити основні демографічні характеристики, звички та преференції цільової аудиторії, а також вивчити їхні потреби та проблеми, пов'язані з організацією власного часу.

Проведення дослідження цільової аудиторії має кілька важливих цілей. По-перше, це дозволяє зрозуміти, хто саме буде використовувати ваш веб-застосунок, і побудувати його таким чином, щоб задовольняти їхні потреби. По-друге, дослідження допомагає виявити основні проблеми та виклики, з якими зіштовхуються цільові користувачі, і знайти ефективні рішення для їх вирішення. Нарешті, дослідження цільової аудиторії допомагає побудувати емпатію з користувачами, що сприяє покращенню користувацького досвіду та впровадженню успішного веб-застосунку.

Для проведення дослідження цільової аудиторії можна використовувати різноманітні методи, такі як опитування, фокус-групи, інтерв'ю, аналіз відгуків та коментарів. Важливо врахувати, що дослідження цільової аудиторії є постійним процесом, оскільки потреби та вимоги користувачів можуть змінюватися з часом. Результати дослідження мають бути використані для прийняття інформованих рішень щодо розробки та вдосконалення веб-

застосунку, забезпечуючи задоволення потреб цільової аудиторії та досягнення успіху проекту.

Основні аспекти дослідження цільової аудиторії для веб-застосунку:

1. Демографічні характеристики: Важливо визначити основні демографічні характеристики вашої цільової аудиторії, такі як вік, стать, місце проживання, освіта, професія тощо. Це допоможе вам краще зрозуміти, які групи людей можуть зацікавитися вашим застосунком.

2. Потреби та проблеми: Вивчіть потреби та проблеми вашої цільової аудиторії, пов'язані з організацією власного часу. Наприклад, це можуть бути люди, які шукають ефективні інструменти для планування та управління своїми завданнями, покращення продуктивності або зменшення стресу.

3. Звички та преференції: Вивчіть звички та преференції вашої цільової аудиторії. Це може включати їхній стиль життя, використання технологій, переваги щодо інтерфейсу користувача, прийняття мобільних пристроїв тощо. Це допоможе вам створити зручний та привабливий для них дизайн та функціонал.

4. Конкурентне середовище: Вивчіть конкурентне середовище існуючих веб-застосунків для організації власного часу. Дослідіть їхні переваги, недоліки та відгуки користувачів. Це допоможе вам знайти свої унікальні переваги.

2.1. Прототипування та подальші процеси створення дизайну

Прототипування є наступним кроком у процесі створення дизайну веб-застосунку. Це дозволяє перевірити і валідувати концепції та функціонал, перш ніж розпочинати розробку на повну шкалу. Прототипування допомагає виявити потенційні проблеми та вдосконалити користувацький досвід перед тим, як інвестувати значні ресурси в програмування та розробку.

Основні кроки прототипування включають:

1.Збір вимог: Зрозуміти функціонал, який має бути включений у додаток, та потреби користувачів.

2.Створення графічних скетчів: Створення першого наближення до вигляду і компонування елементів інтерфейсу.

3.Інтерактивність: Додавання взаємодії, яка дозволяє користувачам взаємодіяти з прототипом.

4.Тестування та зворотній зв'язок: Проведення тестів з потенційними користувачами, щоб отримати їхній відгук та зробити необхідні зміни у прототипі.

Після прототипування, коли концепція та функціонал додатку вже валідовані, розпочинається створення дизайну. Основні кроки створення дизайну веб-застосунку включають:

1. Вибір кольорової палітри та типографії: Створення гармонійного вигляду, який відповідає бренду та цільовій аудиторії.
2. Створення інтерфейсу: Розміщення елементів інтерфейсу (кнопки, поля введення, меню тощо) у зручний та естетичний спосіб.
3. Розробка стилів та компонентів: Визначення стилів для різних елементів і їх повторне використання для забезпечення консистентності дизайну.
4. Розробка інтерактивності: Додавання анімацій, переходів та інших взаємодійних ефектів, щоб поліпшити користувацький досвід.
5. Тестування та оптимізація: Проведення тестів дизайну з реальними користувачами, збір їхнього відгуку та вносити необхідні зміни для оптимізації дизайну.

Усі ці кроки спрямовані на створення ефективного та привабливого дизайну веб-застосунку, який задовольняє потреби та очікування цільової аудиторії.

2.3. JavaScript та база даних

JavaScript є однією з найпопулярніших мов програмування для створення веб-застосунків. Вона використовується для реалізації динамічних ефектів на сторінках веб-сайтів, а також для розробки повноцінних веб-додатків. JavaScript підтримується всіма сучасними браузерами і дозволяє розробникам створювати інтерактивні застосунки з високим рівнем функціональності. Саме тому її обрали для реалізації веб застосунку.

JavaScript має численні переваги порівняно з іншими мовами програмування.

Розглянемо переваги:

1. Інтерактивність та динамічність: JavaScript дозволяє створювати інтерактивні веб-сторінки, що реагують на дії користувача. З його допомогою можна змінювати та оновлювати вміст сторінок без перезавантаження, створювати анімацію, додавати валідацію форм, реалізовувати перехід між сторінками та багато іншого. Це дає можливість створювати більш ефективний та привабливий користувацький досвід.
2. Широке застосування: JavaScript використовується не тільки у веб-розробці, але і у багатьох інших сферах, таких як мобільна розробка (з використанням фреймворків, таких як React Native та Ionic), розробка настільних додатків (з використанням Electron), розробка серверного коду (з використанням Node.js), розробка ігор та багато іншого. Це робить JavaScript універсальною мовою програмування, що дозволяє розробникам використовувати свої навички на різних платформах та в різних сферах.
3. Велика кількість ресурсів та підтримка спільноти: JavaScript має велику кількість ресурсів, документації та онлайн-курсів, що допомагають початківцям вивчити мову та покращити свої навички. Також JavaScript має

активну та велику спільноту розробників, яка надає підтримку, спільно працює над вдосконаленням мови та створенням нових інструментів.

4. Наявність фреймворків та бібліотек: JavaScript має багато популярних фреймворків та бібліотек, таких як React, Angular, Vue.js, що спрощують розробку веб-додатків та забезпечують високу продуктивність та швидкість розробки. Вони надають готові рішення для роботи зі станом додатку, маршрутизації, створення компонентів, роботи з API та багато іншого.

5. Сумісність з браузерами: JavaScript є однією з основних мов програмування, яку підтримують всі сучасні браузери. Це означає, що веб-додатки, написані на JavaScript, працюватимуть на будь-якому сучасному пристрої та браузері без необхідності встановлення додаткового програмного забезпечення.

6. Розширюваність та екосистема: JavaScript має багато розширень та плагінів, які дозволяють розширювати його можливості та пристосовувати під конкретні потреби розробки. Крім того, існує велика кількість сторонніх бібліотек та модулів, які полегшують розробку та допомагають швидко вирішувати різноманітні завдання.

7. Швидкість та продуктивність: JavaScript виконується безпосередньо у браузері, що дозволяє досягати високої швидкості та продуктивності веб-додатків. Завдяки оптимізаціям та покращенням у реалізації JavaScript-двигуна, таких як V8 (використовується в браузері Google Chrome), швидкість виконання JavaScript-коду значно зросла.

База даних є важливою складовою будь-якого веб-застосунку, що потребує зберігання та обробки даних. База даних дозволяє зберігати дані в структурованому форматі, забезпечує доступ до цих даних, а також забезпечує можливість здійснювати різноманітні операції з даними, такі як пошук, фільтрація, сортування тощо.

У даному розділі буде розглянуто використання JavaScript та бази даних для реалізації функціоналу веб-застосунку для організації власного часу. Буде описано основні принципи роботи з JavaScript та базою даних, а також будуть розглянуті конкретні інструменти та технології, що використовуються для розробки веб-додатків.

JavaScript

JavaScript є однією з найпопулярніших мов програмування у світі, яка знаходить своє застосування у веб-розробці, розробці мобільних додатків, гейм-девелопменті, створенні додатків для роботів та багато іншого. JavaScript є мовою програмування, що виконується на боці клієнта (front-end) та сервера (back-end). Його основна функція полягає в тому, щоб надати можливість розробникам створювати динамічні та інтерактивні веб-сторінки.

JavaScript має безліч фреймворків та бібліотек, що роблять його розробку значно простішою та швидшою. Найпопулярнішими з них є React, Angular, Vue.js, jQuery, Node.js та багато інших. JavaScript є мовою програмування з відкритим кодом, що означає, що будь-який розробник може доповнити її новими функціями та покращеннями. Це сприяє активному розвитку та постійному оновленню мови.

React NodeJS

React та Node.js - це дві популярні технології, які використовуються для розробки веб-додатків. React - це JavaScript-бібліотека для побудови інтерфейсів користувача, що дозволяє розробникам легко створювати високопродуктивні та інтерактивні веб-додатки. Використання React забезпечує швидкість розробки завдяки використанню компонентного підходу, який дозволяє створювати малі, повторно використовувані та легко змінювані компоненти.

Node.js, з іншого боку, є середовищем виконання JavaScript, яке дозволяє розробникам використовувати JavaScript для розробки серверної частини веб-додатків. Node.js забезпечує високу продуктивність і можливість обробки

багатьох запитів одночасно завдяки використанню необхідних технологій та бібліотек.

Використання React та Node.js разом дозволяє розробникам створювати повнофункціональні веб-додатки з високою продуктивністю та швидкістю роботи, які можуть працювати на різних платформах і пристроях. Крім того, React та Node.js забезпечують широкий спектр інструментів та бібліотек, що дозволяють розробникам створювати більш складні додатки зі складними функціями та можливостями.

MongoDB

MongoDB - це документ-орієнтована система управління базами даних (NoSQL), яка зберігає дані у вигляді документів у форматі BSON (Binary JSON), який є бінарним представленням JSON-об'єктів. MongoDB дозволяє зберігати дані в будь-якій структурі без задання строго схеми даних, що дозволяє легко маніпулювати даними з високою швидкістю.

MongoDB підтримує горизонтальне масштабування, тобто можливість додавати нові сервери для збільшення потужності системи без переносу даних на нові машини. MongoDB також підтримує реплікацію, що дозволяє створювати дублюючі копії даних на інших серверах для забезпечення високої доступності.

MongoDB дозволяє легко виконувати запити до даних, використовуючи мову запитів MongoDB, яка нагадує мову запитів SQL, але має деякі відмінності, зокрема, дозволяє працювати з документами в форматі JSON. MongoDB може бути використана для зберігання різноманітних даних, таких як зображення, відео, звуки, текстові файли, які можуть бути збережені в документах в форматі BSON.

MongoDB є досить популярною базою даних в середовищі веб-розробки, оскільки вона дозволяє зберігати дані у вигляді документів, що надає більшу гнучкість та швидкість розробки, особливо коли мається справа зі зберіганням невеликої кількості даних різного типу, що має змінну структуру.

Material-UI

Material-UI - це відкрита бібліотека компонентів React, що реалізує дизайн Google Material Design. Вона надає розробникам можливість створювати якісний і сучасний веб-інтерфейс за допомогою вбудованих готових компонентів.

Material-UI дозволяє швидко та просто створювати високоякісні, модерні та гнучкі веб-додатки, не витрачаючи багато часу на дизайн і розробку власних компонентів. Бібліотека включає в себе широкий спектр компонентів, таких як кнопки, текстові поля, таблиці, модальні вікна, меню та інші. Кожен компонент можна налаштувати за допомогою властивостей, що дає можливість відтворити більш точно задуманий дизайн.

Material-UI підтримує React Hooks, що дозволяє забезпечити більш просту та зрозумілу логіку взаємодії компонентів. Також Material-UI має вбудовану систему тем, що дозволяє налаштувати кольорову гаму та інші параметри для всього додатку або окремих компонентів.

У цілому, Material-UI є потужним інструментом для створення привабливого та сучасного веб-інтерфейсу за короткий час. Бібліотека підтримується активною комуніті та має велику кількість ресурсів для навчання та документації.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

Для створення веб застосунку необхідно пройти всі етапи створення дизайну та розробити fron-end частину. Перший етап дослідити цільову аудиторію , далі створити прототип майбутнього сайту та розробка коду. Для цього використовуємо такі програми як Figma (для створення прототипу дизайну) та Visual Studio Code (для розробки коду).

3.1. Дослідження цільової аудиторії

Провівши дослідження ось які данні та статистику отримали:

1.Демографічні характеристики:

- Віковий розподіл цільової аудиторії:

  18-24 роки: 25%

  25-34 роки: 40%

  35-44 роки: 20%

  45-54 роки: 10%

  55+ років: 5%

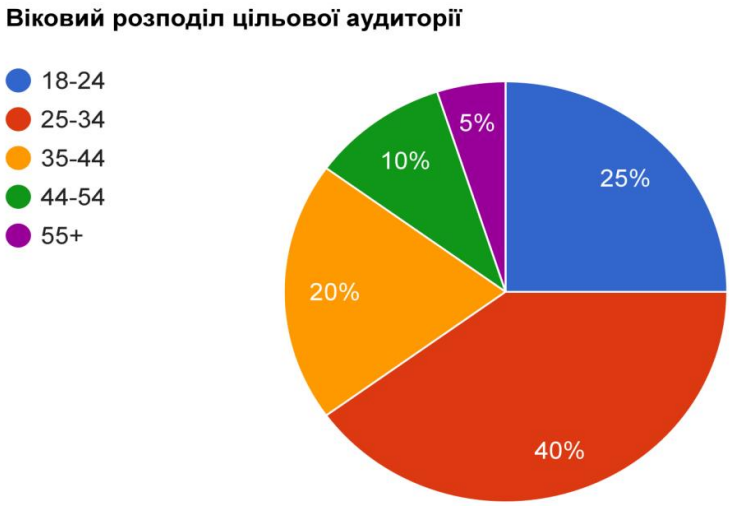


Рисунок 3.1.1 Віковий розподіл цільової аудиторії

- Стать:

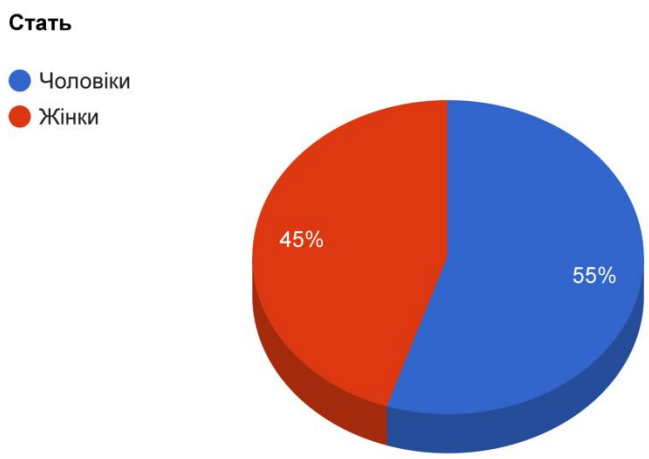


Рисунок 3.1.2 Стать

- Місце проживання:

  Місто: 70%

  Сільська місцевість: 30%

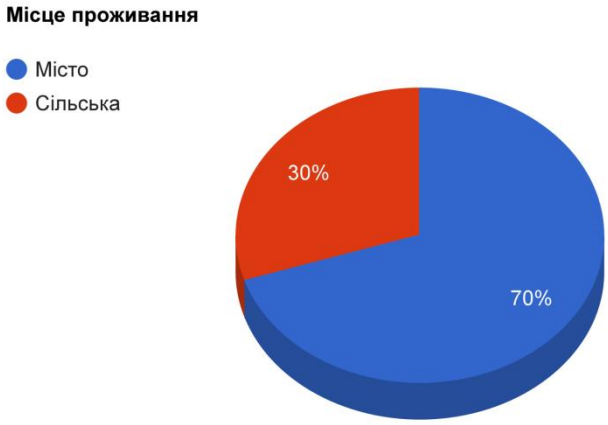


Рисунок 3.1.3 Місце проживання

- Освіта:

Вища: 60%

Середня: 30%

Незакінчена середня: 10%

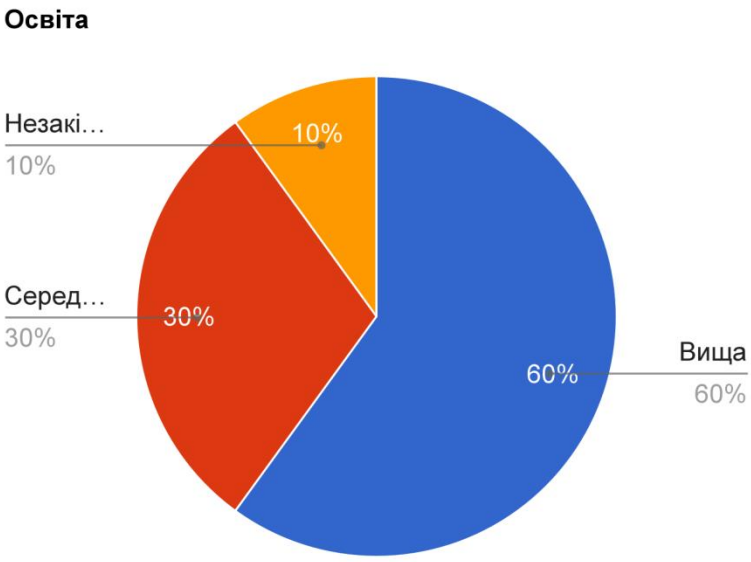


Рисунок 3.1.4 Освіта

2. Потреби та проблеми:

Основні потреби цільової аудиторії:

- Ефективне планування та організація робочих завдань: 70%
- Управління особистими термінами та дедлайнами: 60%
- Покращення продуктивності та зосередженості: 55%
- Зменшення стресу та покращення роботи-життя балансу: 50%

Основні проблеми, з якими стикаються цільові користувачі:

- Недостатня організація робочого часу: 45%
- Втрати часу на непродуктивні завдання: 40%
- Важкість управління пріоритетами: 35%
- Недостатня мотивація та самодисципліна: 30%

3. Звички та преференції:

Звички цільової аудиторії:

- Використання смартфонів: 90%
- Використання комп'ютерів: 75%
- Використання планшетів: 40%

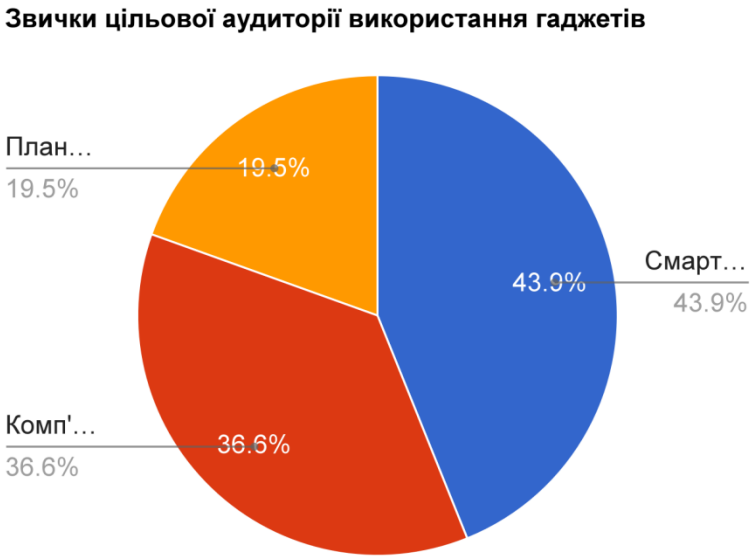


Рисунок 3.1.5 Звички цільової аудиторії

Преференції щодо інтерфейсу користувача:

- Мінімалістичний та інтуїтивно зрозумілий дизайн: 70%
- Можливість налаштування та персоналізації: 60%
- Мобільна доступність: 80%
- Інтеграція з календарем та популярними сервісами: 50%

3.2.Створення макету прототипу програми

Прототип показує призначені функції та кнопки, їх розташування та маркування, а також загальну форму пристрою, але жодна з кнопок не працює. Цей тип прототипу достатньо для дослідження сценаріїв використання та прийняття рішення щодо наприклад, чи відповідні зображення кнопок і мітки функцій достатньо [3].

Прототипування є важливим кроком у процесі створення дизайну веб-застосунку. Це дозволяє перевірити і валідувати концепції та функціонал, перш ніж розпочинати розробку на повну шкалу. Прототипування допомагає виявити потенційні проблеми та вдосконалити користувацький досвід перед тим, як інвестувати значні ресурси в програмування та розробку.

Макет прототипу програми розроблений у Figma має вигляд представлений на рисунку 3.2.1.

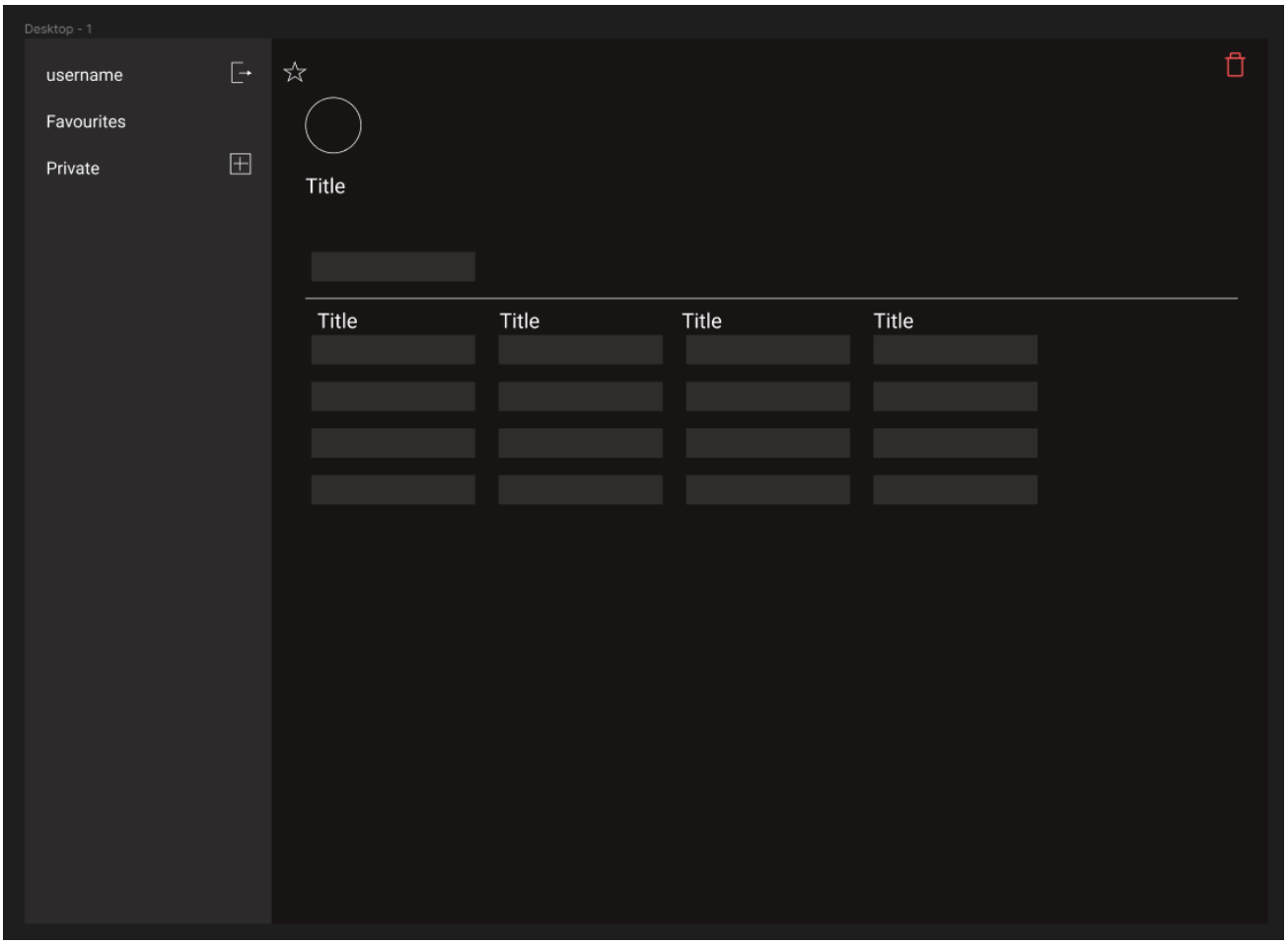


Рисунок 3.2.1 Прототип

Основні кроки прототипування включають:

1. Збір вимог: Зрозуміти функціонал, який має бути включений у додаток, та потреби користувачів.
2. Створення графічних скетчів: Створення першого наближення до вигляду і компонування елементів інтерфейсу.
3. Інтерактивність: Додавання взаємодії, яка дозволяє користувачам взаємодіяти з прототипом.
4. Тестування та зворотній зв'язок: Проведення тестів з потенційними користувачами, щоб отримати їхній відгук та зробити необхідні зміни у прототипі.

Після прототипування, коли концепція та функціонал додатку вже валідовані, розпочинається створення дизайну. Основні кроки створення дизайну веб-застосунку включають:

- Вибір кольорової палітри та типографії: Створення гармонійного вигляду, який відповідає бренду та цільовій аудиторії.
- Створення інтерфейсу: Розміщення елементів інтерфейсу (кнопки, поля введення, меню тощо) у зручний та естетичний спосіб.
- Розробка стилів та компонентів: Визначення стилів для різних елементів і їх повторне використання для забезпечення консистентності дизайну.
- Розробка інтерактивності: Додавання анімацій, переходів та інших взаємодійних ефектів, щоб поліпшити користувацький досвід.
- Тестування та оптимізація: Проведення тестів дизайну з реальними користувачами, збір їхнього відгуку та вносити необхідні зміни для оптимізації дизайну.

Усі ці кроки спрямовані на створення ефективного та привабливого дизайну веб-застосунку, який задовольняє потреби та очікування цільової аудиторії.

3.3.Фронтенд розробка

Структура проекту має вигляд представлений на рисунку 3.2.2.:

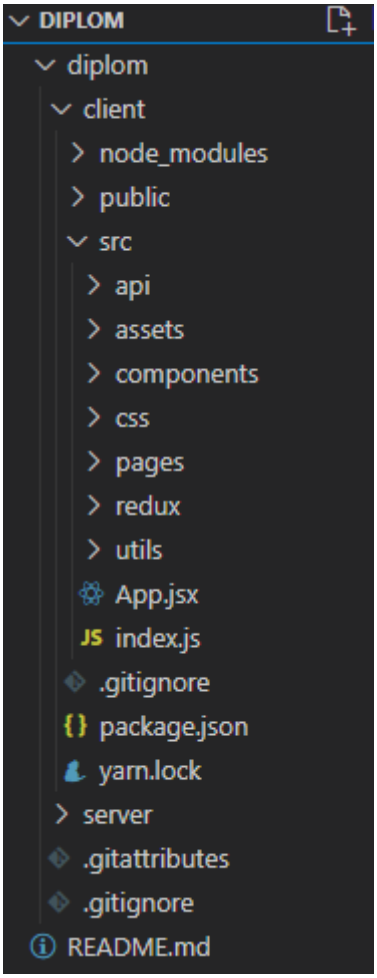


Рисунок 3.2.2 Прототип

Поділяється структура на дві частини клієнтську та серверну.

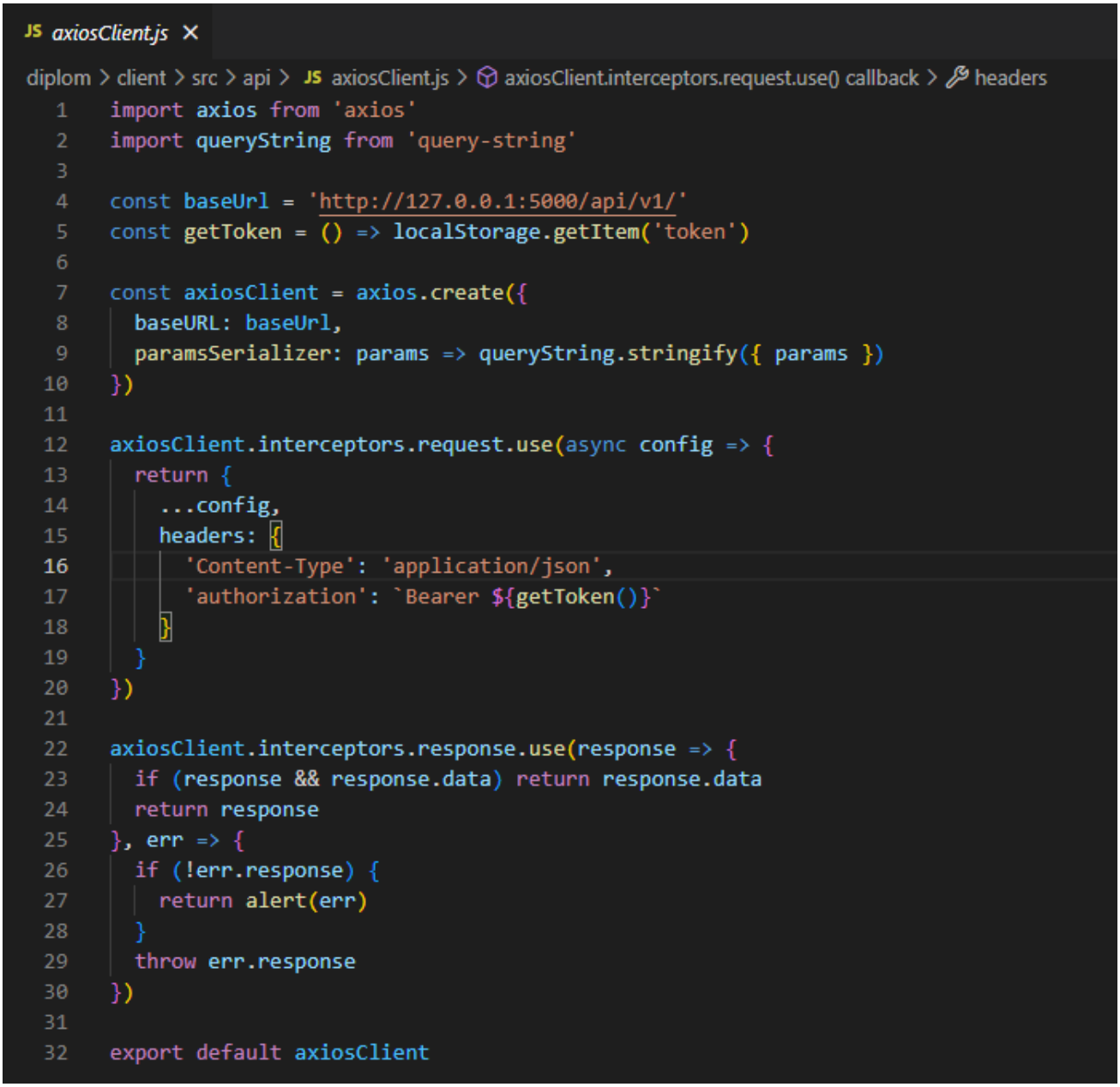
Для взаємодії з серверним API написано код використовуючи бібліотеку axios.

authApi.js

diplom > client > src > api > authApi.js > ...

```
1  import axiosClient from "./axiosClient"
2
3  const authApi = {
4    signup: params => axiosClient.post('auth/signup', params),
5    login: params => axiosClient.post('auth/login', params),
6    verifyToken: () => axiosClient.post('auth/verify-token')
7  }
8
9  export default authApi
10
```

Далі було написано модуль axiosClient, який створює клієнт axios для взаємодії з сервером а також налаштувати заголовки запитів та обробляти відповіді сервера.



Наступний модуль дозволяє зручно виконувати різні операції з дошками на сервері, використовуючи визначені функції, які використовують клієнт axiosClient для взаємодії з сервером.

JS boardApi.js

diplom > client > src > api > JS boardApi.js > default

```
import axiosClient from './axiosClient'

const boardApi = {
  create: () => axiosClient.post('boards'),
  getAll: () => axiosClient.get('boards'),
  updatePositoin: (params) => axiosClient.put('boards', params),
  getOne: (id) => axiosClient.get(`boards/${id}`),
  delete: (id) => axiosClient.delete(`boards/${id}`),
  update: (id, params) => axiosClient.put(`boards/${id}`, params),
  getFavourites: () => axiosClient.get('boards/favourites'),
  updateFavouritePosition: (params) => axiosClient.put('boards/favourites', params)
}

export default boardApi
```

Цей модуль дозволяє зручно виконувати різні операції з секціями дошки на сервері, використовуючи визначені функції, які використовують клієнт axiosClient для взаємодії з сервером.

JS sectionApi.js

diplom > client > src > api > JS sectionApi.js > default

```
import axiosClient from './axiosClient'

const sectionApi = {
  create: (boardId) => axiosClient.post(`boards/${boardId}/sections`),
  update: (boardId, sectionId, params) => axiosClient.put(
    `boards/${boardId}/sections/${sectionId}`,
    params
  ),
  delete: (boardId, sectionId) => axiosClient.delete(`boards/${boardId}/sections/${sectionId}`)
}

export default sectionApi
```

Останній модуль дозволяє зручно виконувати різні операції з завданнями на дошці на сервері, використовуючи визначені функції, які використовують клієнт axiosClient для взаємодії з сервером.

taskApi.js

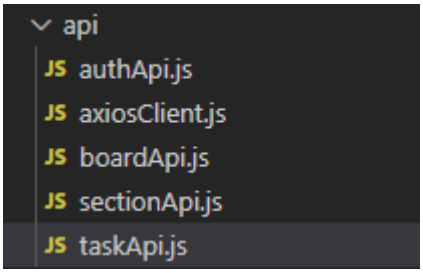
diplom > client > src > api > taskApi.js > ...

```
import axiosClient from './axiosClient'

const taskApi = {
  create: (boardId, params) => axiosClient.post(`boards/${boardId}/tasks`, params),
  updatePosition: (boardId, params) => axiosClient.put(
    `boards/${boardId}/tasks/update-position`,
    params
  ),
  delete: (boardId, taskId) => axiosClient.delete(`boards/${boardId}/tasks/${taskId}`),
  update: (boardId, taskId, params) => axiosClient.put(
    `boards/${boardId}/tasks/${taskId}`,
    params
  )
}

export default taskApi
```

В кінцевому результаті отримаємо таку структуру файлів по API



Цей код є головним компонентом App, який використовує багато бібліотек та компонентів для налаштування теми, маршрутизації та компонентів розміщення.

App.jsx

diplom > client > src > App.jsx > App

```jsx
import '@fontsource/roboto/300.css'
import '@fontsource/roboto/400.css'
import '@fontsource/roboto/500.css'
import '@fontsource/roboto/700.css'
import './css/custom-scrollbar.css'
import CssBaseLine from '@mui/material/CssBaseline'
import { ThemeProvider, createTheme } from '@mui/material/styles'
import { BrowserRouter, Route, Routes } from 'react-router-dom'
import AppLayout from './components/layout/AppLayout'
import AuthLayout from './components/layout/AuthLayout'
import Home from './pages/Home'
import Board from './pages/Board'
import Signup from './pages/Signup'
import Login from './pages/Login'

function App() {
  const theme = createTheme({
    palette: { mode: 'dark' }
  })

  return (
    <ThemeProvider theme={theme}>
      <CssBaseLine />
      <BrowserRouter>
        <Routes>
          <Route path='/' element={<AuthLayout />}>
            <Route path='login' element={<Login />} />
            <Route path='signup' element={<Signup />} />
          </Route>
          <Route path='/' element={<AppLayout />}>
            <Route index element={<Home />} />
            <Route path='boards' element={<Home />} />
            <Route path='boards/:boardId' element={<Board />} />
          </Route>
        </Routes>
      </BrowserRouter>
    </ThemeProvider>
  );
}

export default App;

```

Додатково пропишемо окремий код який буде відповідати за рендеринг головного компонента App за допомогою бібліотеки React.

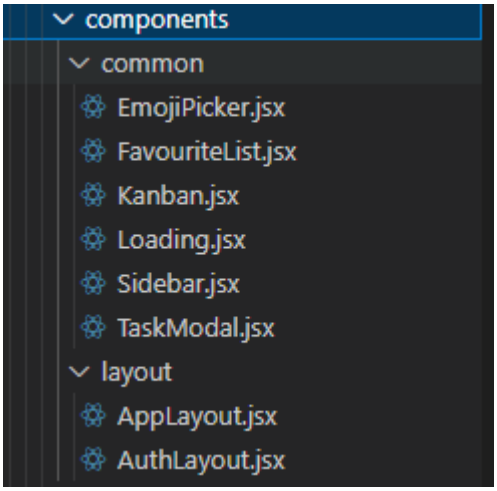
index.js

diplom > client > src > index.js > ...

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { store } from './redux/store'
import { Provider } from 'react-redux';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Provider store={store}>
    <App />
  </Provider>
);
```

Компоненти поділяються на дві частини common та layout , ось їх структура :



Розглянемо код кожного компонента.

EmojiPicker.jsx

Основна функціональність цього компонента полягає в виборі емодзі з емодзі-пікера та передачі вибраного емодзі до батьківського компонента через props.onChange при зміні емодзі.

Компонент містить стан для зберігання вибраного емодзі (selectedEmoji) та стану для відображення/приховування емодзі-пікера (isShowPicker).

Використовується useEffect для оновлення вибраного емодзі при зміні значення props.icon.

У компоненті рендериться контейнер Box, який містить Typography для відображення вибраного емодзі та Box, який містить емодзі-пікер. Відображення/приховування емодзі-пікера залежить від значення стану isShowPicker. Компонент Picker постачається з бібліотеки emoji-mart і використовується для відображення емодзі-пікера.

Цей компонент можна використовувати в інших компонентах, передаючи необхідні значення через props.

EmojiPicker.jsx

diplom > client > src > components > common > EmojiPicker.jsx > EmojiPicker

```jsx
import { Box, Typography } from '@mui/material'
import React, { useState, useEffect } from 'react'
import { Picker } from 'emoji-mart'

import 'emoji-mart/css/emoji-mart.css'

const EmojiPicker = props => {
  const [selectedEmoji, setSelectedEmoji] = useState()
  const [isShowPicker, setIsShowPicker] = useState(false)

  useEffect(() => {
    setSelectedEmoji(props.icon)
  }, [props.icon])

  const selectEmoji = (e) => {
    const sym = e.unified.split('-')
    let codesArray = []
    sym.forEach(el => codesArray.push('0x' + el))
    const emoji = String.fromCodePoint(...codesArray)
    setIsShowPicker(false)
    props.onChange(emoji)
  }

  const showPicker = () => setIsShowPicker(!isShowPicker)

  return (
    <Box sx={{ position: 'relative', width: 'max-content' }}>
      <Typography
        variant='h3'
        fontWeight='700'
        sx={{ cursor: 'pointer' }}
        onClick={showPicker}
      >
        {selectedEmoji}
      </Typography>
      <Box sx={{
        display: isShowPicker ? 'block' : 'none',
        position: 'absolute',
        top: '100%',
        zIndex: '9999'
      }}>
        <Picker theme='dark' onSelect={selectEmoji} showPreview={false} />
      </Box>
    </Box>
  )
}

export default EmojiPicker
```

FavouriteList.jsx

Цей компонент відображає список обраних (улюблених) пунктів та надає можливість їх перетягування для зміни порядку.

Loading.jsx може використовуватися для показування процесу завантаження або очікування, поки деякі дані завантажуються або обробляються.

Loading.jsx

diplom > client > src > components > common > Loading.jsx > default

```jsx
import { Box, CircularProgress } from '@mui/material'

const Loading = props => {
  return (
    <Box sx={{
      display: 'flex',
      alignItems: 'center',
      justifyContent: 'center',
      width: '100%',
      height: props.fullHeight ? '100vh' : '100%'
    }}>
      <CircularProgress />
    </Box>
  )
}

export default Loading
```

Sidebar.jsx є боковим меню веб-додатка. Він містить список дошок, облікові дані користувача та можливість додавати нові дошки.

TaskModal.jsx цей компонент приймає вхідні параметри boardId, task, onUpdate та onClose. Він зберігає стан задачі (task), назви (title) та вмісту (content) за допомогою хуків useState.

При зміні параметра task, компонент оновлює свій стан і встановлює відповідні значення для title та content. Крім того, він також виконує функцію updateEditorHeight, яка оновлює висоту редактора.

У компоненті є функції onClose, deleteTask, updateTitle та updateContent, які виконують відповідні дії при натисканні кнопок або оновленні полів вводу.

Коли компонент TaskModal рендериться, він відображає модальне вікно (Modal), яке містить різні елементи, такі як кнопка видалення, поле вводу назви, дата створення, роздільник та редактор задачі (CKEditor).

Усередині компонента використовуються референції (useRef) для отримання доступу до елементів DOM та зміни їх властивостей.

У другій частині layout є лише два файли.

AppLayout

AppLayout.jsx 1

diplom > client > src > components > layout > AppLayout.jsx > AppLayout > useEffect() callback

```jsx
import { Box } from '@mui/material'
import { useState, useEffect } from 'react'
import { Outlet, useNavigate } from 'react-router-dom'
import { useDispatch } from 'react-redux'
import authUtils from '../../utils/authUtils'
import Loading from '../common/Loading'
import Sidebar from '../common/Sidebar'
import { setUser } from '../../redux/features/userSlice'

const AppLayout = () => {
  const navigate = useNavigate()
  const dispatch = useDispatch()
  const [loading, setLoading] = useState(true)

  useEffect(() => {
    const checkAuth = async () => {
      const user = await authUtils.isAuthenticated()
      if (!user) {
        navigate('/login')
      } else {
        // save user
        dispatch(setUser(user))
        setLoading(false)
      }
    }
    checkAuth()
  }, [navigate])

  return (
    loading ? (
      <Loading fullHeight />
    ) : (
      <Box sx={{
        display: 'flex'
      }}>
        <Sidebar />
        <Box sx={{
          flexGrow: 1,
          p: 1,
          width: 'max-content'
        }}>
          <Outlet />
        </Box>
      </Box>
    )
  )
}

export default AppLayout
```

AuthLayout

AuthLayout.jsx

diplom > client > src > components > layout > AuthLayout.jsx > AuthLayout

```jsx
import { Container, Box } from '@mui/material'
import { useState, useEffect } from 'react'
import { Outlet, useNavigate } from 'react-router-dom'
import authUtils from '../../utils/authUtils'
import Loading from '../common/Loading'
import assets from '../../assets'

const AuthLayout = () => {
  const navigate = useNavigate()
  const [loading, setLoading] = useState(true)

  useEffect(() => {
    const checkAuth = async () => {
      const isAuth = await authUtils.isAuthenticated()
      if (!isAuth) {
        setLoading(false)
      } else {
        navigate('/')
      }
    }
    checkAuth()
  }, [navigate])

  return (
    loading ? (
      <Loading fullHeight />
    ) : (
      <Container component='main' maxWidth='xs'>
        <Box sx={{
          marginTop: 8,
          display: 'flex',
          alignItems: 'center',
          flexDirection: 'column'
        }}>
          <img src={assets.images.logoDark} style={{ width: '100px' }} alt='app logo' />
          <Outlet />
        </Box>
      </Container>
    )
  )
}

export default AuthLayout
```

Стилі

custom-editor.css

diplom > client > src > css > # custom-editor.css > .ck.ck-editor__main>.ck-editor__editable

```css
.ck.ck-editor__main>.ck-editor__editable {
  background-color: transparent;
  border: none;
  outline: none;
}

.ck.ck-editor__main>.ck-editor__editable.ck-focused {
  border: none;
}

.ck.ck-toolbar .ck.ck-toolbar__separator {
  background-color: transparent;
}

.ck.ck-toolbar.ck-toolbar_grouping {
  background-color: transparent;
  border: none;
}

.ck.ck-button,
.ck.ck-button.ck-on {
  color: white;
  background-color: transparent;
}

.ck.ck-button:hover,
.ck.ck-button.ck-off:hover,
.ck.ck-button.ck-on:hover {
  background-color: transparent !important;
  cursor: pointer;
  color: gray;
}

.ck-dropdown__panel .ck.ck-button.ck-off.ck-button__with-text {
  background-color: black !important;
}

.ck.ck-list__item {
  background-color: black !important;
}

.ck-dropdown__panel .ck-dropdown__panel-visible {
  border: none;
}

.ck-editor__top {
  position: sticky !important;
  top: 0 !important;
  background: #121212 !important;
```

custom-scrollbar.css

diplom > client > src > css > # custom-scrollbar.css > ::-webkit-scrollbar

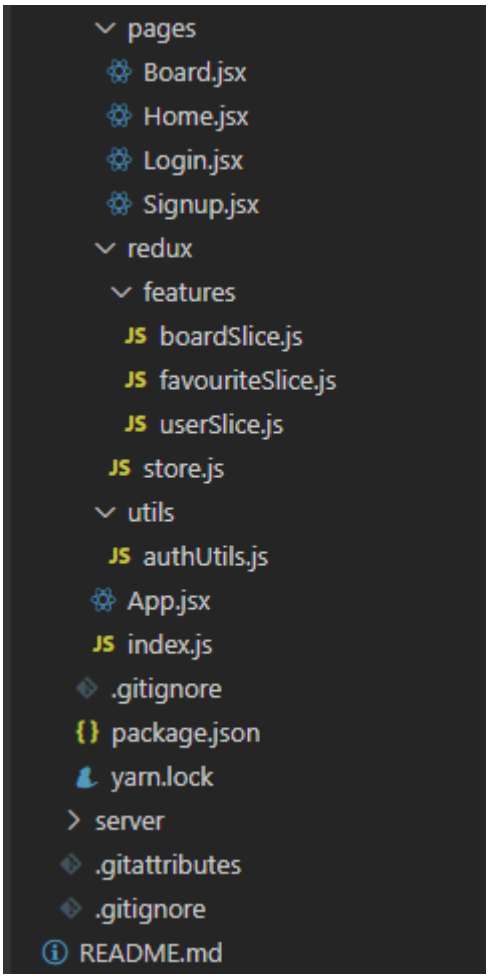
```css
::-webkit-scrollbar {
  width: 5px;
  height: 5px;
}

::-webkit-scrollbar-track {
  background: gray;
}

::-webkit-scrollbar-thumb {
  background-color: darkslategray;
}

::-webkit-scrollbar-thumb:hover {
  background-color: #555;
}
```

А також решта файлів



ВИСНОВОК

Ця дипломна робота присвячена розробці системи управління завданнями для організації робочих процесів. Робота передбачає розробку веб-додатка, який надає можливість створювати, редагувати та видаляти завдання, а також керувати розкладом та пріоритетом завдань.

У дипломній роботі використовуються сучасні технології розробки веб-додатків, зокрема React та Material-UI для фронтенду, а також Node.js і Express.js для бекенду. У роботі також використовується база даних MongoDB для зберігання даних про завдання та користувачів.

Головні функціональні можливості додатка включають створення завдань з вказанням заголовку, опису та інших важливих атрибутів, встановлення термінів виконання, визначення пріоритету завдань та їх фільтрацію за різними критеріями. Додаток також надає можливість організації завдань у вигляді дошки, де вони можуть бути груповані за проектами або категоріями.

Однією з ключових особливостей розробленої системи є використання реактивного підходу до оновлення даних, що дозволяє користувачам бачити оновлення в режимі реального часу без необхідності оновлювати сторінку.

Дипломна робота включає детальний опис проекту, аналіз вимог, проектування системи, розробку функціональних модулів та їх тестування. Результатом роботи є повноцінний веб-додаток, який може бути використаний для ефективного керування завданнями та покращення робочих процесів в організації.

У цілому, ця дипломна робота демонструє глибоке розуміння сучасних технологій розробки веб-додатків та здатність до розробки функціональної та ефективної системи управління завданнями. Результати цієї роботи можуть бути корисними для організацій, які шукають ефективні інструменти для керування завданнями та підвищення продуктивності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Idea Digital Agency. Як визначити цільову аудиторію? 2021. – [Електронний ресурс]. – Режим доступу: https://ideadigital.agency/blog/yak-viznachiti-tsilovu-auditoriyu/ (дата звернення: 15.04.2023).
2. Крокфорд, Д. JavaScript: The Good Parts. O'Reilly Media. 2008. - 174 с
3. Preece, J., Rogers, Y., & Sharp, H. (2015). Дизайн взаємодії: Поза взаємодією людини і комп'ютера. Видавництво "Дія". 2015. - 819 ст.
4. Круг, С. Не змушуйте мене думати: Поведінкова евристика і дизайн веб-сайтів. Київ: Видавництво "Київський Університет". 2014.
5. Офіційна документація React – [Електронний ресурс]. – Режим доступу: https://reactjs.org/docs/ (дата звернення: 15.04.2023).
6. Офіційна документація Redux – [Електронний ресурс]. – Режим доступу: https://redux.js.org/ (дата звернення: 15.04.2023).
7. Офіційна документація React Router – [Електронний ресурс]. – Режим доступу: https://reactrouter.com/ (дата звернення: 15.04.2023).
8. Сайт Material-UI – [Електронний ресурс]. – Режим доступу: https://mui.com/ (дата звернення: 15.04.2023).
9. Allen D. Getting Things Done: The Art of Stress-Free Productivity. - New York: Penguin Books, 2015. - 352 p.
10. Forsyth P. Successful Time Management. - Kogan Page, 2018. - 192 p.
11. Krug S. Don’t Make Me Think: A Common Sense Approach to Web Usability. - New Riders, 2014. - 216 p.
12. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. - Sebastopol: O’Reilly Media, 2020. - 310 p.
13. Tilkov S., Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs. - IEEE Internet Computing, 2019. - Vol. 23(6). - P. 80-83.

14. Chodorow K. MongoDB: The Definitive Guide. - Sebastopol: O’Reilly Media, 2019. - 416 p.
15. Brown E. Web Development with Node and Express. - Sebastopol: O’Reilly Media, 2019. - 350 p.