ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Web-застосунку менеджер задач мовою JavaScript з використанням фреймворку React»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Євгеній ФЕЩЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-44

Євгеній ФЕЩЕНКО

Керівник: Тимур ДОВЖЕНКО
к.т.н.

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Фещенку Євгенію Вікторовичу

1. Тема кваліфікаційної роботи: «Розробка веб-додатку менеджер задач мовою програмування JavaScript з використанням фреймворку React»
керівник кваліфікаційної роботи к.т.н., доцент кафедри ІПЗ Тимур ДОВЖЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про веб-додатки, види веб-додатків, відмінності веб-застосунку та веб-сайту, огляд і аналіз існуючих аналогів та технологій, вибір оптимальних засобів для реалізації веб-додатку, використання JavaScript з React для клієнтської сторони, Node.js з Express для серверної частини, забезпечення зручного використання на різних пристроях та тестування перед запуском.

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

1. Огляд та аналіз існуючих Web-застосунків для управління задачами. визначити переваги та недоліки.
2. Визначити функціональні та нефункціональні вимоги до Web-застосунку.

3. Проектування архітектури та структури Web-застосунку, враховуючи обрані технології та інструменти.
4. Розробка серверної та клієнтської частини Web-застосунку з використанням мови JavaScript та бібліотеки React.
5. Провести тестування Web-застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до додатку.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Файлова структура проєкту.
6. Діаграма класів.
7. Схема бази даних.
8. Мапа сайту.
9. Екранні форми.
10. Відео роботи веб-застосунку.
11. Апробація результатів дослідження.

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Проектування архітектури та структури Web-застосунку	14.03-20.03.2024	
4	Розробка серверної та клієнтської частини Web-застосунку	21.03-20.04.2024	
5	Тестування Web-застосунку	21.04-28.04.2024	
6	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
7	Розробка демонстраційних матеріалів	06.05-12.05.2024	
8	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Євгеній ФЕЩЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Тимур ДОВЖЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 46 стор., 17 рис., 14 джерел.

Мета роботи – спрощення процесу управління задачами за допомогою Web-застосунку створеного мовою JavaScript з використанням фреймворку React.

Об'єкт дослідження – процес управління задачами.

Предмет дослідження – Web-застосунок для управління задачами.

Короткий зміст роботи: В роботі було проведено аналіз задач, необхідних для виконання на веб-платформі в результаті чого були отримані функціональні та нефункціональні вимоги. Також було проведено аналіз наявних аналогів, визначені позитивні й негативні сторони кожного аналогу. Створений додаток менеджера задач складається з клієнтської та серверної сторони.

Розроблено веб-додаток «Task Manager» та програмно реалізовані ключові функціональні можливості, зокрема: реєстрація та авторизація, редагування профілю користувача, створення, редагування та видалення дошок, створення, редагування та видалення колонок та карток на дошці, перетягування карток по дошці, фільтрування та сортування карток за міткою, можливість вибрати тему веб-застосунку, створення, редагування та видалення працівників. В роботі використано мову програмування JavaScript для програмування логіки додатку, фреймворк React для створення користувацького інтерфейсу, NodeJS для створення серверної сторони додатка та інші технології.

Сферою використання застосунку є ПрАТ «Вайдманн-МПФ».

КЛЮЧОВІ СЛОВА: JAVASCRIPT, REACT, NODEJS, ВЕБ-ЗАСТОСУНОК, МЕНЕДЖЕР ЗАДАЧ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HTTP – HyperText Transfer Protocol — «протокол передачі гипертексту»

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

DOM – Document Object Model

SPA – Single Page Application

MPA – Multi-Page Application

SaaS – Software as a Service

JWT – JSON Web Token

ВСТУП

Веб-застосунки стали необхідною частиною нашого життя. Вони дозволяють нам взаємодіяти з інформацією, спілкуватися, працювати та розважатися. У сучасному світі, де інформаційний потік постійно зростає, організація та планування завдань стають критично важливими для продуктивності. Веб-застосунки для управління задачами є незамінним інструментом. Менеджери задач допомагають нам організовувати робочий процес, відстежувати завдання та планувати ресурси.

Ця дипломна робота ставить за мету розробити веб-застосунок менеджера задач на базі бібліотеки React, використовуючи мову JavaScript. Для досягнення цієї мети буде проведено аналіз існуючих рішень, вибір додаткових інструментів, безпосередньо розробка додатка з урахуванням потреб компанії «Вайдманн-МПФ», та фінальне тестування для забезпечення його працездатності та зручності використання.

Дослідження буде зосереджено на розробці веб-застосунку менеджера задач з використанням бібліотеки React на JavaScript. В якості методів дослідження будуть застосовані аналіз літератури, порівняльний аналіз, проектування, програмування та тестування.

Очікується, розроблений веб-застосунок на базі фреймворку React для створення продуктивного, зручного та функціонального менеджера задач. Практична значущість полягає в можливості використовувати для організації та управління задачами в корпоративному середовищі.

Результатом цієї дипломної роботи є розроблений веб-застосунок менеджера задач на базі фреймворку React. Застосунок буде мати детальний опис його функціональних можливостей, архітектури та дизайну.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття та класифікація «Веб-застосунок» та «Веб-сайт»

У сучасному цифровому світі веб-сайти та веб-застосунки стали невід'ємною частиною нашого повсякденного життя. Вони використовуються для отримання інформації, спілкування, розваг, роботи та багато чого іншого.

Веб-сайт - це колекція веб-сторінок, пов'язаних між собою, які розміщені на веб-сервері та доступні за допомогою веб-браузера. Він має унікальну URL-адресу, яка використовується для його доступу. Веб-сайти зазвичай статичні, тобто їхній вміст не змінюється динамічно у відповідь на дії користувача [1].

Серед основних характеристики веб-сайтів:

Статичний контент: Веб-сайти зазвичай містять статичний контент, такий як текст, зображення та відео. Цей контент не змінюється динамічно у відповідь на дії користувача.

Інформаційна мета: Веб-сайти в основному використовуються для надання інформації користувачам. Вони можуть служити джерелом новин, освітнім ресурсом, онлайн-магазином або просто візитною карткою для компанії чи особи.

Одностороння комунікація: Веб-сайти зазвичай пропонують односторонню комунікацію між веб-сайтом та користувачем. Користувачі можуть читати контент, переглядати зображення та дивитися відео, але вони не можуть динамічно взаємодіяти з ним.

Приклади веб-сайтів:

- **https://www.bbc.com/**
- **https://www.amazon.com/**
- **https://nv.ua/**
- **https://www.president.gov.ua/**

Веб-застосунок - це програмне забезпечення, яке працює в веб-браузері та доступне через Інтернет. Він має подібний інтерфейс та функціональність, як і традиційний десктопний додаток, але не потребує встановлення на комп'ютер користувача. Веб-застосунки динамічні, тобто їхній вміст може змінюватися в залежності від дій користувача. Веб-застосунки використовуються в різних сферах, включаючи бізнес, освіту, розваги та багато інших. Вони можуть бути простими або складними, і їх функціональність може бути різноманітною [2].

Основні характеристики веб-застосунків:

- **Динамічний контент:** Веб-застосунки генерують динамічний контент у відповідь на дії користувача. Це дозволяє користувачам взаємодіяти з веб-додатком, вводити дані та отримувати персоналізовані результати.
- **Інтерактивні можливості:** Веб-застосунки пропонують широкий спектр інтерактивних можливостей, таких як форми, опитування, чати та онлайн-ігри. Це дозволяє користувачам активно брати участь у роботі веб-застосунку.
- **Збір даних:** Веб-застосунки можуть збирати дані про користувачів, такі як їхні дії, вподобання та демографічну інформацію. Ці дані можуть використовуватися для покращення роботи веб-застосунку, персоналізації контенту та цільової реклами.
- **Синхронізація:** Деякі веб-застосунки можуть синхронізувати дані на різних пристроях, що дозволяє користувачам отримувати доступ до своєї інформації з будь-якого місця.

Приклади веб-застосунків:

- **https://mail.google.com/mail/**
- **https://www.google.com/docs/about/**
- **https://www.facebook.com/**
- **https://www.instagram.com/**
- **https://accounts.ukr.net/login?lang=ru**

1.2 Основні відмінності «Веб-застосунку» та «Веб-сайту».

Хоча терміни «веб-сайт» і «веб-застосунок» часто використовуються взаємозамінно, між ними є суттєві відмінності. Розуміння цих відмінностей допоможе вам вибрати правильний інструмент для своїх потреб.

Веб-сайти та веб-програми, доступні через веб-браузер, пропонують принципово різний досвід користувача. Веб-сайт представляє інформацію в зрозумілий і візуально привабливий спосіб, використовуючи текст, зображення та відео. Ви можете переглядати цю інформацію у власному темпі, дізнаваючись про послуги компанії, читаючи новинні статті або досліджуючи портфоліо фотографа.

Веб-сайти чудово демонструють вміст у стилі односторонньої комунікації, тобто ви споживаєте інформацію, але не можете безпосередньо її змінювати. Це робить їх ідеальними для ситуацій, коли ви хочете встановити присутність в Інтернеті, поділитися інформацією про компанію або надати оновлення новин широкій аудиторії. Оскільки веб-сайти переважно статичні, їх, як правило, легше та дешевше створювати та підтримувати порівняно з веб-застосунками.

З іншого боку, веб-застосунки — це як програмне забезпечення, до якого можна отримати доступ через веб-браузер, усуваючи потребу у завантаженні чи інсталяції. Веб-програми оживають завдяки вашій взаємодії. Ви можете вводити дані, завантажувати файли, співпрацювати з іншими та маніпулювати інформацією в режимі реального часу. Веб-застосунок може бути службою веб-пошти, як-от Gmail, де ви можете надсилати й отримувати електронні листи, або потужним інструментом для редагування фотографій, як-от Canva, де ви можете редагувати свої фотографії безпосередньо у веб-переглядачі. Такі інструменти управління проектами, як Trello або платформи онлайн-банкінгу, також є яскравими прикладами веб-додатку. Оскільки веб-додатки динамічні та реагують на введення користувача, їх складніше розробляти та підтримувати порівняно з веб-сайтами.

Вибір між веб-сайтом і веб-застосунком залежить від цілі. Якщо ваша головна мета — представити інформацію в зрозумілій і привабливій формі, веб-сайт — ідеальне рішення. Це дозволяє вам продемонструвати свій вміст, створити ідентичність свого бренду та охопити широку аудиторію. Але якщо у вас є бачення більш інтерактивного досвіду, де користувачі можуть активно брати участь і виконувати завдання, тоді потрібен веб-додаток. Веб-застосунки надають користувачам платформу для створення, редагування та спільної роботи, сприяючи більш насиченому та динамічнішому досвіду онлайн.

1.3 Види веб-застосунків.

Існує багато способів класифікувати веб-програми, але ось кілька поширених типів на основі їх функціональності та цільової аудиторії:

Статичні веб-застосунки: Це найпростіший тип веб-застосунків. Вони відображають інформацію, яка є попередньою та не змінюється динамічно залежно від взаємодії користувача. Вони часто створюються за допомогою HTML, CSS і JavaScript для базової інтерактивності.

Динамічні веб-програми: Вони можуть обробляти дані користувача, генерувати персоналізовані відповіді та оновлювати вміст у режимі реального часу. Для обробки інформації та взаємодії з базами даних вони покладаються на серверні мови програмування, такі як PHP, Python або Java. Прикладами можуть бути: служба електронної пошти Gmail, платформа онлайн-покупок Amazon або соціальна мережа Facebook.

Односторінкові програми (SPA): Тип динамічної веб-програми, де більша частина взаємодії відбувається на одній веб-сторінці. Вміст динамічно оновлюється без перезавантаження всієї сторінки, що забезпечує більш зручну роботу користувача. Популяризовані такими фреймворками, як React.js або Angular, SPA часто використовуються для веб-застосунків, які більше схожі на

нативні додатки. Приклади включають Gmail, Trello (інструмент управління проектами) або багато сучасних платформ соціальних мереж.

Багатосторінкові програми (MPA): Це традиційні веб-програми, де користувачі переміщуються між кількома сторінками HTML для доступу до різних функцій. Кожна сторінка є окремою сутністю і вимагає повного перезавантаження під час перемикання між ними. Вони все ще широко використовуються для складних веб-додатків із великою кількістю інформації чи функцій.

Прогресивні веб-додатки (PWA): Гібрид традиційних веб-програм і мобільних програм. Вони пропонують такі функції, як офлайн-доступ, push-сповіщення та встановлення на головному екрані, стираючи межі між веб-програмами та нативними програмами. PWA добре працюють на будь-якому пристрої та можуть бути хорошою альтернативою розробці окремих мобільних програм. Приклади PWA: веб-плеєр Spotify та Pinterest.

Програми електронної комерції: Веб-застосунки, спеціально розроблені для здійснення покупок в Інтернеті. Вони дозволяють користувачам переглядати продукти, додавати товари до кошиків, здійснювати безпечні онлайн-платежі та відстежувати свої замовлення. Платформи електронної комерції часто інтегруються з платіжними шлюзами, системами управління запасами та службами доставки. Приклади включають магазини Amazon, eBay, Etsy або Shopify.

Бізнес веб-застосунки: Веб-застосунки, які використовуються компаніями для керування внутрішніми операціями, відносинами з клієнтами або робочими процесами проекту. Вони можуть бути розроблені спеціально для конкретних потреб або використовувати рішення програмного забезпечення як послуги (SaaS), які задовольняють різні бізнес-функції. Приклади включають програмне забезпечення CRM (Customer Relationship Management), системи планування ресурсів підприємства (ERP) або інструменти управління проектами.

Веб-ігри: Інтерактивні ігри, у які можна грати повністю у веб-браузері. Вони можуть варіюватися від простих ігор-головоломок до складних ігор для кількох гравців. Веб-ігри можуть використовувати передові графічні та аудіотехнології, щоб забезпечити захоплюючий ігровий досвід.

Системи керування контентом (CMS): Веб-застосунки, які спрощують створення вмісту веб-сайту та керування ним. Вони дозволяють користувачам, які можуть не мати досвіду програмування, редагувати текст, завантажувати зображення та публікувати вміст на веб-сайті. Популярні платформи CMS включають WordPress, Joomla, Drupal або Wix.

2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ, ІНСТРУМЕНТІВ ТА ТЕХНОЛОГІЙ

2.1 Аналіз існуючих рішень.

Менеджер задач можна реалізувати по-різному. Наприклад у вигляді веб-застосунку, який буде доступний через браузер на будь-якому пристрої, мобільного додатку для смартфонів та планшетів, або ж декстопному, тобто для використання потрібно попередньо інсталювати на персональний комп'ютер.

Також застосунки можна поділити за функціональністю. Простий менеджер задач в якому реалізовані базові функції для створення та редагування задач. Розширений для використання додаткових функцій, наприклад календарі, звіті, аналітика тощо.

Застосунки такого типу може використовувати як одна людина так і декілька одночасно, тобто їх можна поділити за аудиторією. Особисті менеджери задач, командний для організації спільної роботи, для бізнезу з розширеними функціями, для управління проектами, командами, ресурсами тощо.

Розглянемо реальні приклади менеджера задач:

Todoist

Todoist — менеджер завдань, який дозволяє створити список завдань і керувати ними різними способами: через веб-сайт, розширення браузера, а також програми для Windows, macOS, Android, iOS. Платформа підтримує повну автоматичну синхронізацію даних між усіма підключеними пристроями, так як вони зберігаються не тільки на пристроях, але і на сервері Todoist. Менеджер дозволяє вільно керувати параметрами кожної конкретної задачі, встановлювати для неї різні проміжні та остаточні терміни, повторні дати, виставляти та змінювати пріоритети, створювати підпроекти та ін. За необхідності весь функціонал сервісу Todoist може бути доступний не тільки онлайн, але й офлайн.

Todoist пропонує дуже широкі можливості для командної роботи, дозволяючи розподіляти завдання та відповідальність між різними учасниками проєкту, а також додавати необмежену кількість людей у робочу групу, що займається виконанням цих чи інших завдань [3].

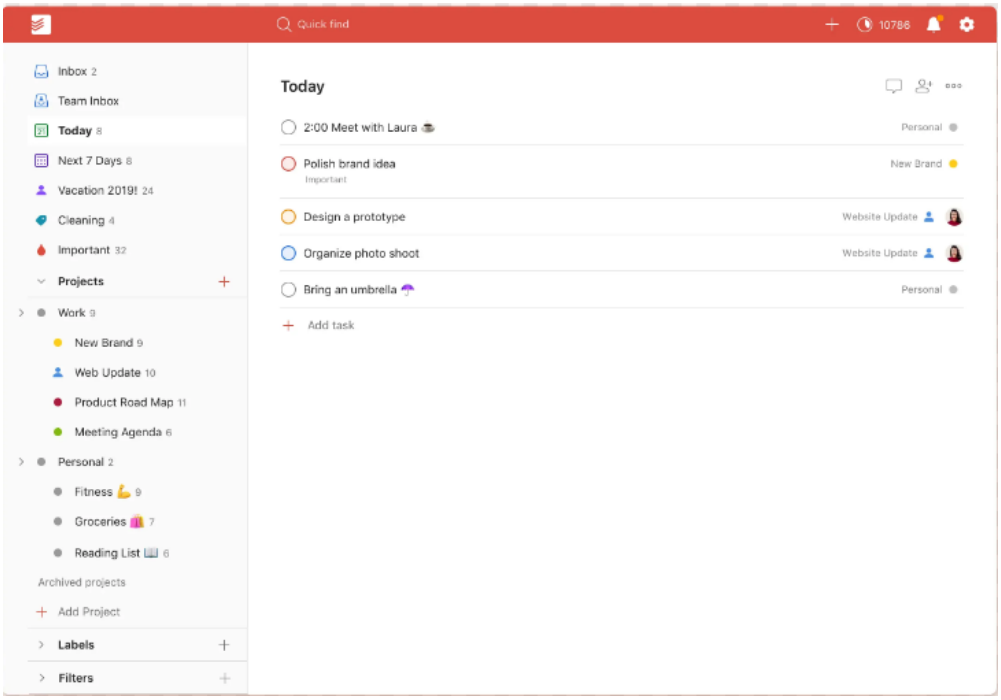


Рис. 2.1 Інтерфейс додатку Todoist

Todoist підтримує 13 комп'ютерних та мобільних платформ, пропонує онлайн-синхронізацію і резервне копіювання, а також інтегрується з Google Calendar та іншими сервісами. Всі акаунти Todoist мають обмеження: безкоштовні акаунти обмежені 80 проєктами та 5 користувачами в кожному, тоді як преміум і бізнес акаунти дозволяють до 200 проєктів і до 50 користувачів в кожному проєкті. Ці обмеження не застосовуються до заархівованих проєктів. Завдання можна організовувати в проєктах, сортувати за фільтрами, присвоювати їм мітки, редагувати, експортувати, призначати відповідальних, а також додавати сповіщення і нагадування.

Додаток є командний, розширений, так як можна працювати групами з розширеним функціоналом, підтримується на будь якій платформі, також існує у вигляді розширення для Google Chrome, яке зображено на рисинку 2.2.

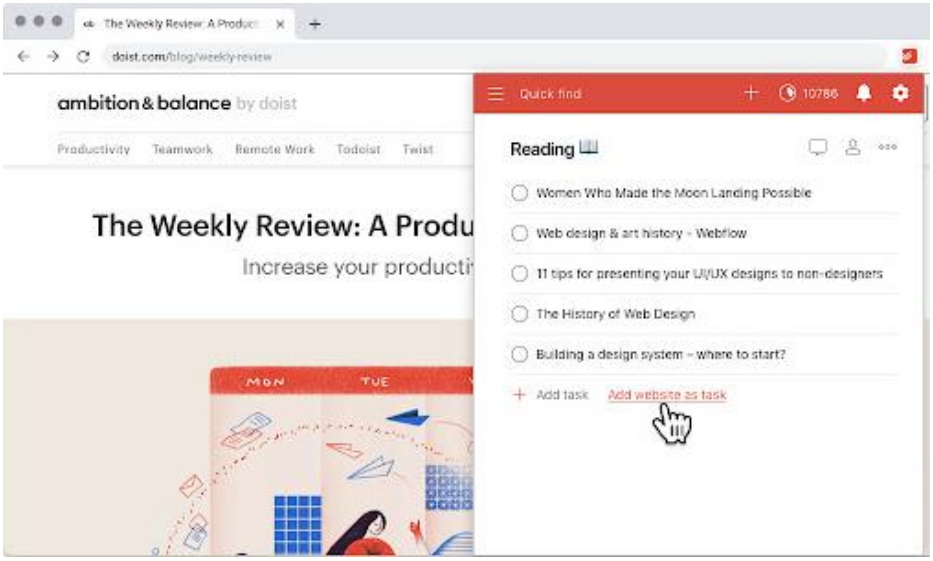


Рис. 2.2 Todoist у вигляді розширення для Google Chrome

У застосунку можна як планувати справи, так і записувати їх. Також він має безліч інструментів. Наприклад планування проектів, встановлення пріоритетів задач, мітки для розподілу і групуванню схожих задач, можна до задачі залишити опис, залишити коментар а також розбити справу на підпункти та багато інших, що робить програму досить універсальною в напрямку менеджер задач.

Також в ньому легко працювати через зручний інтерфейс. Поріг входження в програму досить інтуїтивний та зрозумілий.

Trello

Trello - це безкоштовна багатоплатформна система управління проектами та завданнями, яка допомагає організувати та візуалізувати роботу. Вона використовує канбан-дошки, списки та картки, щоб чітко структурувати завдання, пріоритети та хід виконання проектів [4].

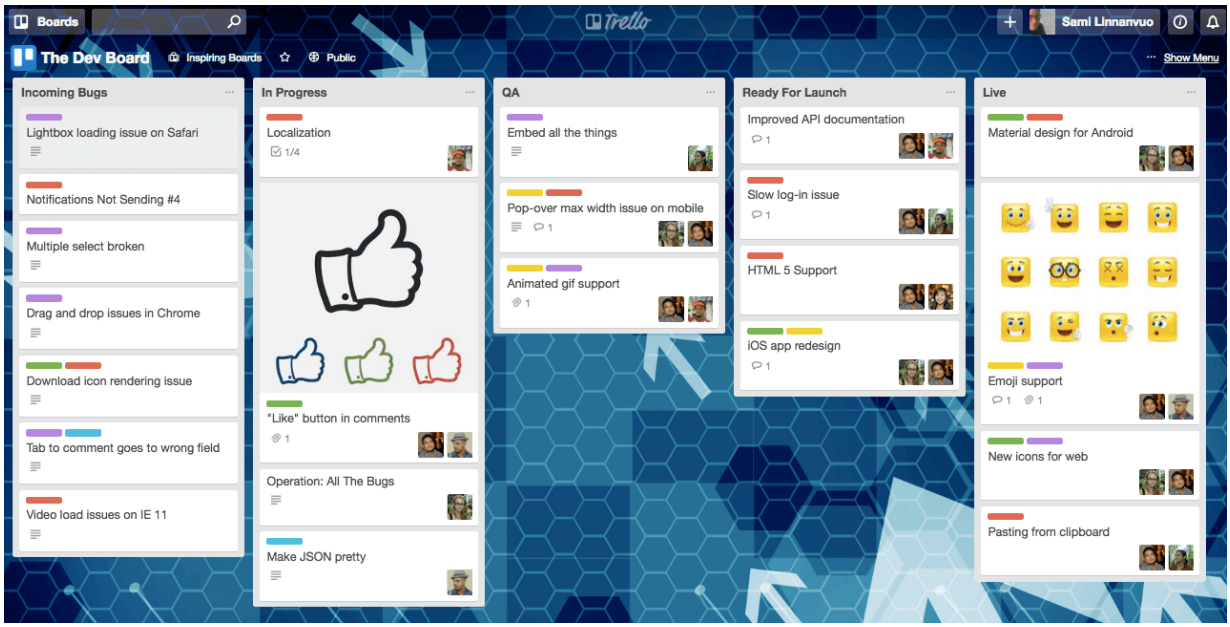


Рис. 2.3 Інтерфейс додатку Trello

Основні функції Trello:

Дошки - створення окремих дошок для різних проектів, команд або тем, налаштування фонового зображення та назви дошки для кращої візуалізації, додавання опису дошки, щоб чітко окреслити її мету та призначення.

Списки - Розбивка дошок на списки, які описують етапи проекту, категорії завдань або робочі процеси, визначення назви та кольору для кожного списку, встановлення обмежень на роботу в списку, щоб уникнути перевантаження.

Картки - створення окремих карток для кожного завдання з описом, дедлайном, учасниками та іншими деталями, додавання заголовка, опису та контрольного списку до карток, призначення етикеток до карток для категоризації та візуалізації, завантаження файлів, зображень та посилань до карток.

Перетягування - переміщення карток між списками та дошками для візуального відображення прогресу, використання клавіатурних скорочень для швидкого переміщення карток.

Призначення - додавання членів команди до карток, щоб вони могли отримувати сповіщення та брати участь у роботі, згадування членів команди в коментарях та описі карток.

Списки справ - створення контрольних списків завдань, розбивка великих завдань на менші, більш керовані підзавдання, відзначення виконаних підзавдань для візуального відстеження прогресу.

Вкладення -додавання файлів, зображень та посилань до карток, зберігання всіх пов'язаних з завданням документів та ресурсів в одному місці, спільний доступ до файлів з членами команди.

Коментарі - обговорення завдань та співпраця з командою в коментарях до карток, згадування членів команди та додавання коментарів до підзавдань, відстеження історії обговорень у кожній картці.

Повідомлення - отримання сповіщень про активність у проектах, таких як нові картки, коментарі та згадки, налаштування сповіщень, щоб отримувати лише те, що вам важливо, інтеграція з Slack та іншими інструментами для отримання сповіщень.

Інтеграції - з'єднання Trello з іншими інструментами, такими як Slack, Google Calendar та Jira, розширення можливостей Trello за допомогою інтеграцій з сторонніми сервісами, автоматизація завдань та робочих процесів за допомогою інтеграцій.

Автоматизація - використання правил автоматизації для виконання повторюваних завдань, створення автоматичних тригерів, які запускають дії, наприклад, надсилання сповіщень або переміщення карток, економія часу та підвищення продуктивності за допомогою автоматизації.

Jira

Jira — це веб-програма, розроблена Atlassian, яка функціонує як інструмент управління проектами. Він особливо добре підходить для гнучких команд розробників програмного забезпечення, але його універсальність дозволяє використовувати його для різних завдань управління проектами в різних відділах [5]. Основні функції:

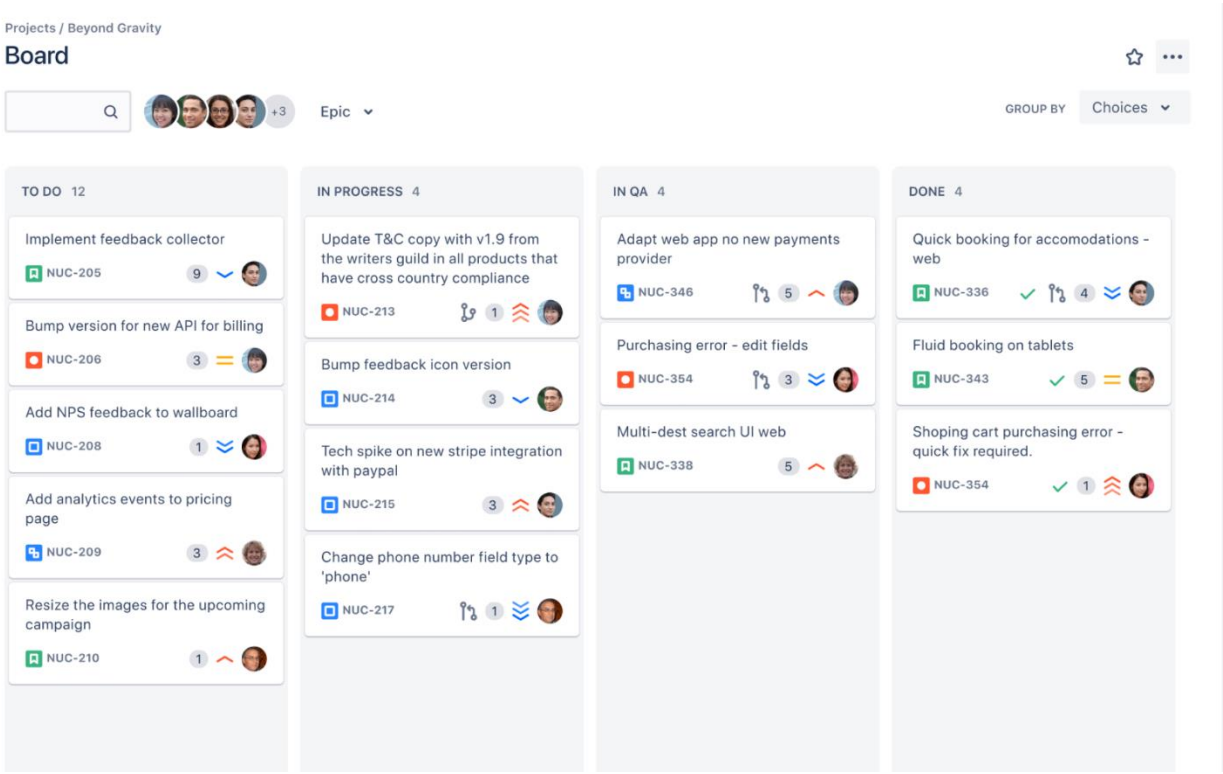


Рис. 2.4 Інтерфейс додатку Jira

Відстеження задач: Jira відмінно справляється з відстеженням проблем, помилок і завдань. Ви можете створювати звіти про проблеми, призначати їх членам команди, відстежувати прогрес і керувати термінами.

Управління проектами: Jira допомагає організувати проекти в робочі процеси з настроюваними етапами. Це дозволяє командам візуалізувати хід проекту та забезпечити плавний перебіг завдань на різних етапах.

Підтримка Agile: Jira пропонує функції, спеціально розроблені для гнучких методологій, таких як Scrum і Kanban. Ви можете створювати історії користувачів, керувати спринтами та відстежувати прогрес у своєму гнучкому робочому процесі.

Автоматизація робочого процесу: Jira дозволяє автоматизувати повторювані завдання у ваших робочих процесах. Це може значно підвищити ефективність і зменшити ручну роботу.

Jira використовують:

Команди розробки програмного забезпечення: широко використовується групами розробників програмного забезпечення для відстеження помилок, гнучкого керування проектами та планування спринтів.

Команди управління проектами: Менеджери проектів у різних відділах можуть використовувати Jira для загального керування завданнями, організації робочого процесу та командної співпраці.

Бізнес-команди: Команди маркетингу, кадрів і операцій можуть використовувати Jira для планування проектів, відстеження проблем і керування робочим процесом.

По суті, Jira — це потужний інструмент управління проектами, який може оптимізувати робочі процеси, покращити співпрацю та покращити видимість проекту для різних команд і проектів.

2.2 Аналітичний огляд інструментів та технологій для розробки веб-застосунку.

2.2.1 Hypertext Markup Language.

HTML або Hypertext Markup Language – це код, який використовується для створення веб-сторінок і веб-додатків. HTML був створений у всесвітній мережі наприкінці 1980-х і на початку 1990-х років.

HTML використовує різноманітні теги для визначення різних елементів, які складають веб-сторінку. Ці теги діють як інструкції для веб-браузера, повідомляючи йому, як відображати вміст певним чином. Заголовки, абзаци, списки, зображення, відео тощо – у HTML є теги для всіх цих елементів, які гарантують, що веб-браузер правильно організовує їх на екрані.

HTML забезпечує тільки структуру, він не підтримує візуальний дизайн чи інтерактивність. За це відповідають каскадні таблиці стилів (CSS) і Javascript відповідно. HTML зосереджується на самому вмісті, описуючи те, що знаходиться на сторінці, у спосіб, який можуть зрозуміти пошукові системи та веб-браузери.

Також HTML це основа для SEO оскільки пошукові системи значною мірою покладаються на HTML для розуміння інформації на веб-сайті, він відіграє вирішальну роль в пошуковій оптимізації (SEO). Правильне використання тегів HTML гарантує, що пошукові системи зможуть точно індексувати та ранжувати вміст веб-сайту.

HTML 5 є найбільш відповідним стандартом для цієї мови розмітки сьогодні. За цим критерієм базується найбільша кількість веб-сайтів в Інтернеті. Ми отримуємо підтримку мультимедіа (аудіо та відео) в останній версії HTML 5.

2.2.2 Cascading Style Sheets.

CSS, або Cascading Style Sheets, є стилістом у світі веб-розробки. Це мова, яка визначає спосіб представлення HTML-вмісту на веб-сторінці, перетворюючи основну структуру на візуально привабливу та зручну для користувача сторінку. CSS пропонує ширший спектр можливостей стилю порівняно зі вбудованими стилями HTML. За допомогою CSS можна створювати складніші макети, анімацію та адаптивні елементи дизайну.

CSS дотримується ієрархії каскадних таблиць стилів, де конкретніші стилі мають перевагу над загальнішими. Це дозволяє точно контролювати зовнішній вигляд ваших веб-сторінок.

Таким чином, CSS забезпечує більш ефективний, зручний і потужний спосіб стилізації веб-сторінок порівняно зі вбудованими стилями HTML. Він сприяє більш чистому коду, узгодженому дизайну, швидшому часу завантаження та більшій гнучкості для створення візуально привабливих і зручних веб-сайтів.

Відокремлення презентації (CSS) від вмісту (HTML) зберігає ваш код упорядкованим і зручним для керування. Уявіть собі документ із змішаним вмістом і інструкціями щодо форматування. Розділення їх значно спрощує редагування та оновлення. За допомогою CSS уся інформація про стилі зберігається в окремому файлі CSS, що робить ваш HTML-код чистішим і більш зосередженим на структурі вмісту.

Такий код значно легше підтримувати і оновлювати. Якщо вам потрібно змінити стиль елемента на всьому веб-сайті, вам потрібно лише змінити файл CSS. Це економить час і зменшує ризик помилок порівняно з необхідністю оновлювати стилі в кожному тегу HTML.

Також CSS відкриває можливість адаптивного дизайну. у сучасному світі, де мобільні пристрої в першу чергу, веб-сайти повинні адаптуватися до різних розмірів екрана. CSS забезпечує адаптивний дизайн, дозволяючи вам створювати

макети, які налаштовуються та оптимально відображаються на настільних комп’ютерах, планшетах і смартфонах.

Зменшення розміру веб-сторінки: Зберігаючи стилі в окремому файлі CSS, ви також уникаєте повторення тієї самої інформації про стиль у кожному тегу HTML. Це може призвести до меншого загального розміру файлу для вашої веб-сторінки, що може покращити швидкість завантаження.

Багаторазове використання стилів: у CSS можна створювати багаторазові стилі. Наприклад, загальний стиль для всіх заголовків (від <h1> до <h6>) або стандартний стиль кнопки можна визначити один раз у CSS і застосувати будь-де, де потрібно, у вашому HTML.

Основні компоненти CSS:

Селектори: діють як ідентифікатори, орієнтуючись на певні елементи HTML на веб-сторінці. Ви можете націлити елементи за їхнім ідентифікатором, класом, іменем тегу або навіть комбінацією цих елементів.

Властивості: вони визначають конкретні візуальні аспекти, які потрібно стилізувати. Загальні властивості включають «font-family» (для стилю шрифту), «color» (для кольорів тексту та елементів), «background-color» (для фону елементів) і «margin» та «padding» (для інтервалів навколо елементів).

Значення: вони визначають спосіб застосування властивостей. Наприклад, властивість «font-family» може мати таке значення, як «Arial» або «Times New Roman», щоб визначити стиль шрифту.

2.2.3 Javascript

JavaScript є основою інтерактивності у світі веб-розробки. Це мова програмування, яка вдихає життя у веб-сторінки, дозволяючи їм динамічно реагувати на дії користувача та оновлювати вміст без необхідності повного

перезавантаження сторінки. Будь який веб-сайт без JavaScript був би статичною сторінкою. JavaScript робить веб-сайти інтерактивними та привабливими.

JavaScript може маніпулювати вмістом HTML веб-сторінки, не перезавантажуючи всю сторінку. Це забезпечує зручнішу роботу користувача, наприклад оновлення кошиків для покупок, відображення даних у реальному часі або створення інтерактивних елементів, таких як каруселі зображень.

JavaScript дозволяє реагувати на введення користувача, як-от натискання кнопок, надсилання форм або рухи миші. Це дозволяє використовувати такі функції, як інтерактивні форми, перевірки підтвердження, акордеони та динамічні меню.

Також JavaScript може додавати анімацію та візуальні ефекти до ваших веб-сторінок, роблячи їх більш привабливими та зручними для користувача. Ви можете створювати анімації для переходів сторінок, взаємодії елементів або навіть інтерактивних ігор.

JavaScript може обробляти та маніпулювати даними на стороні клієнта (браузер користувача). Це дозволяє виконувати такі завдання, як перевірка введених користувачем даних, виконання обчислень і локальне сортування чи фільтрування даних.

І однією з важливих частин сучасних веб-додатків це надсилання запитів для отримання даних. JavaScript може асинхронно спілкуватися з серверами, тобто він може отримувати дані або надсилати інформацію на сервер, не перериваючи роботу користувача. Це вмикає такі функції, як живий чат, пропозиції автозаповнення або оновлення вмісту на основі дій користувача.

Навички JavaScript стають все більш цінними для розробників із повним стеком (Fullstack), які працюють як із зовнішнім (інтерфейс користувача), так і з внутрішнім (на стороні сервера) веб-додатків. JavaScript можна використовувати для обох сторін за допомогою таких інструментів, як Node.js, що робить його універсальним навиком для веб-розробки [6].

2.3 Необхідність застосування фреймворку у веб-розробці.

Популярні фреймворки, як React, Angular і Vue.js, побудовані на JavaScript, забезпечують структуровані способи розробки складних та інтерактивних веб-застосунків. Ці фреймворки використовують можливості JavaScript для створення багатофункціонального та динамічного веб-досвіду.

Фреймворки для веб-розробки схожі на готові набори інструментів, які пропонують структурований підхід до створення веб-сайтів і веб-застосунків. Ось чому вони стають дедалі важливішими:

Фреймворки сприяють повторному використанню коду. Компоненти та функції можна повторно використовувати в проекті або навіть у різних проектах. Використовуючи готові компоненти та встановлені шаблони кодування, фреймворки можуть значно прискорити процес розробки. Це дозволяє веб-розробникам швидше виконувати проекти та дотримуватись термінів більш ефективно.

Фреймворки забезпечують структурований і організований спосіб написання коду. Це робить код легшим для розуміння, підтримки та модифікації, особливо для великих проектів, де кілька розробників працюють над однією кодовою базою. обре налагоджені фреймворки постійно оновлюються та підтримуються великими спільнотами. Це гарантує, що код, написаний сьогодні, залишатиметься актуальним і сумісним із майбутніми версіями та технологіями браузерів.

Фреймворки часто розробляються з урахуванням оптимізації продуктивності. Вони можуть містити такі функції, як мінімізація коду, механізми кешування та найкращі методи ефективної обробки даних.

Також багато фреймворків мають такі вбудовані функції, як обробка помилок і перевірка, які можуть допомогти запобігти помилкам і забезпечити якість коду. Це призводить до більш стабільних і надійних веб-додатків.

Популярні фреймворки часто мають чітко визначені шаблони дизайну та найкращі практики, які сприяють узгодженому та зручному для користувача досвіду веб-додатків, створених за допомогою цього фреймворку. Також багато фреймворків мають великі та активні спільноти розробників. Це надає доступ до великої кількості ресурсів, навчальних посібників і форумів, де розробники можуть знайти допомогу, поділитися знаннями та бути в курсі останніх досягнень.

Сторонні бібліотеки та плагіни можуть розширювати функціональність фреймворків. Це дозволяє розробникам легко інтегрувати додаткові функції та функції без необхідності писати все з нуля.

Підсумовуючи, фреймворки веб-розробки пропонують швидший, ефективніший і зручніший спосіб створення веб-додатків. Вони надають структуру, найкращі практики, набір інструментів, та забирають складну роботу на себе, і дають змогу розробникам швидко і зручно створювати високоякісні та масштабовані веб-інтерфейси.

3 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ

3.1 JavaScript.

JavaScript є однією з найпоширеніших мов програмування, яка використовується для розробки клієнтської сторони веб-додатків, на рисунку 3.1 зображено рейтинг популярності мови програмування. Ця мова програмування є динамічною та об'єктно-орієнтованою, це надає розробникам широкі можливості для створення функціональних інтерфейсів та взаємодії з користувачем.

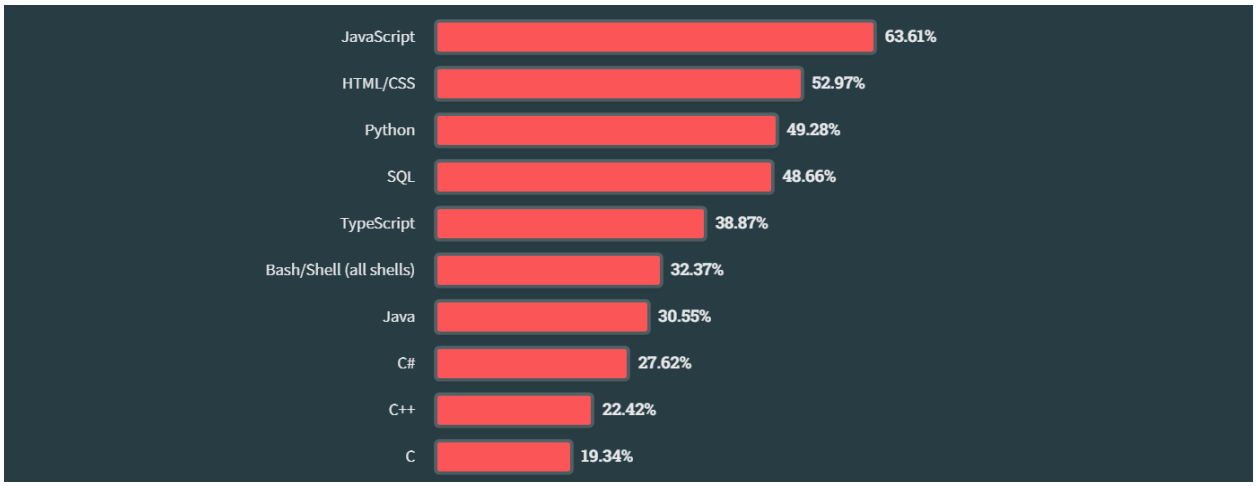


Рис. 3.1 Рейтинг популярності мови програмування серед опитаних на StackOverflow 2023 р.

У контексті клієнтської сторони, JavaScript використовується для динамічного створення та зміни елементів веб-сторінки. Також з його допомогою можна реагувати на дії користувача, обробляти події, валідувати дані та забезпечувати інтерактивність веб-додатків.

Розробники JavaScript проявили значну кмітливість. Замість створення ще однієї мови сценаріїв, вони передбачливо побудували її навколо об'єктної моделі документа (DOM - Document Object Model).

Ця модель розділяє частини HTML-документа на окремі об'єкти, кожен з яких має власні властивості та методи, якими можна керувати за допомогою JavaScript.

JavaScript має велику кількість вбудованих функцій і методів, що спрощують роботу з маніпуляцією DOM-дерева, обробкою форм, валідацією даних, анімацією, взаємодією з сервером та іншими завданнями. Також існують багато сторонніх бібліотек і фреймворків, які розширюють можливості JavaScript та допомагають в розробці більш складних проєктів.

Традиційно JavaScript був королем у світі інтерфейсної веб-розробки, обробляючи те, що користувачі бачать і з чим взаємодіють на веб-сторінці. Але ландшафт змінився! Давайте заглибимося в сферу JavaScript для серверної розробки:

Розвиток Node.js відіграв ключову роль у перенесенні JavaScript на сервер. Node.js — це середовище виконання з відкритим вихідним кодом, яке дозволяє вам виконувати код JavaScript поза браузером, фактично дозволяючи JavaScript використовувати як мову для сторони сервера.

Для розробників, які вже знайомі з JavaScript для Frontend розробки, Node.js дозволяє їм використовувати наявні знання JS як для клієнтських, так і для серверних завдань. Це може оптимізувати процеси розробки та зменшити необхідність перемикання контексту між різними мовами.

Величезна екосистема бібліотек і фреймворків JavaScript також поширюється на серверну частину. Такі популярні фреймворки, як Express.js, забезпечують надійну основу для створення веб-серверів і API за допомогою JavaScript.

3.2 Бібліотека React

ReactJS, розроблений командою Facebook, є провідною бібліотекою JavaScript, широко застосовуваною для створення динамічних користувацьких

інтерфейсів у сучасних веб-застосунках. Виступаючи як основний компонент екосистеми JavaScript, ReactJS пропонує розробникам численні переваги для побудови високоефективних додатків.

Основна перевага ReactJS полягає в його здатності створювати масштабні веб-застосунки, що використовують динамічні дані без потреби в перезавантаженні сторінки. Це досягається завдяки використанню віртуального DOM (Document Object Model), який дозволяє ReactJS вносити зміни до інтерфейсу користувача ефективно, оновлюючи лише ті частини, які були змінені. Це значно підвищує продуктивність і забезпечує плавне та швидке користувацьке враження.

ReactJS базується на концепції компонентів, що дозволяє створювати модульні та повторно використовувані частини інтерфейсу користувача. Компоненти ReactJS є самостійними елементами з власною логікою та розміткою, які легко масштабувати та використовувати в різних частинах додатка. Це сприяє кращій організації коду, знижує дублювання та покращує ефективність розробки.

React потрібен для розробки сучасних інтерфейсів користувача в застосунках. Він дозволяє розбити інтерфейс на компоненти, ефективно керувати станом застосунку, спрощує взаємодію з API та забезпечує швидкий та масштабований реактивний рендерінг.

У даному веб-застосунку ReactJS використовується для розробки клієнтської частини. Використання ReactJS дозволяє додатку легко адаптуватися до нових вимог, масштабуватися і впроваджувати нові функції, включаючи інтерактивні елементи, анімацію та інші складні аспекти користувацького інтерфейсу.

Наприклад, з ReactJS можна створювати динамічні форми введення даних, які автоматично оновлюються відповідно до введених користувачем даних, забезпечуючи відгук у реальному часі та підвищуючи користувацький досвід. Графіки можна будувати за допомогою компонентів ReactJS, що дозволяє

динамічно оновлювати візуалізацію даних відповідно до змін у даних користувача.

Також він має такі переваги:

Використання віртуального DOM та ефективного алгоритму оновлення дозволяє робити мінімальні зміни в реальному DOM, що покращує продуктивність застосунків;

React пропагує односторонній потік даних, що сприяє простоті та передбачуваності управління станом додатків, полегшує відлагодження та тестування застосунків.

Спільнота розробників, що використовує React велика та активна. Є безліч ресурсів, бібліотек та інструментів для розробки. Також існує багато сторонніх бібліотек, які підтримують React і допомагають розширити його функціональність;

ReactJS відіграє ключову роль у створенні сучасних веб-додатків, дозволяючи розробникам будувати потужні, масштабовані та орієнтовані на користувача рішення. Завдяки своїй універсальності, високій продуктивності та модульності, він є ідеальним вибором для розробки продуктивних веб-застосунків [7].

3.3 Допоможні інструменти та бібліотеки для розробки веб-застосунку

3.3.1 Redux Toolkit

Redux, популярна бібліотека JavaScript для управління станом застосунків, особливо в середовищі React, була створена для вирішення проблем управління складними станами в багатокомпонентних веб-застосунках. Redux дозволяє створювати застосунки, де стан є однозначним і передбачуваним, що спрощує процес розробки та підвищує його прозорість.

Одна з головних особливостей Redux полягає в його концепції "єдиного джерела правди". Всі стани додатка зберігаються в одному глобальному об'єкті, який називається "store". Такий підхід дозволяє легко передавати і оновлювати стани додатка. Дії, що змінюють стан, проходять через спеціальні функції, звані "reducers", що дозволяє створити передбачуваний потік даних і забезпечити прогнозованість стану застосунка.

Redux також пропонує можливості для виконання більш складних операцій, таких як асинхронні дії, міжпотокова комунікація та збереження стану. Ці функції роблять Redux універсальним рішенням для застосунків будь-якого розміру, від невеликих до великих і складних систем.

Наприклад, асинхронні дії можуть бути використані для роботи з віддаленими ресурсами, такими як сервери API. Це особливо корисно в сценаріях, де необхідно виконати кілька асинхронних операцій одночасно або в певному порядку. Щодо міжпотокової комунікації, Redux дозволяє розробникам створювати додатки, які ефективно обмінюються даними між різними частинами програми, зберігаючи контроль над потоком даних.

Redux Toolkit (RTK) — це набір офіційних утиліт, призначених для оптимізації роботи з Redux. Сам Redux забезпечує міцну основу для передбачуваного та масштабованого керування станом програми, але RTK спрощує звичайні завдання та пропонує ряд переваг, які підвищують як продуктивність розробника, так і якість програми.

Однією з найважливіших переваг RTK є його здатність спростити налаштування Redux. Redux вимагає налаштування сховища вручну, що може включати шаблонний код і стати громіздким для складних програм. Функція configureStore RTK вирішує цю проблему, пропонуючи спрощений підхід. Ця функція вміло поєднує кілька важливих завдань в одному рядку коду

Middleware — це потужна концепція в Redux, яка дозволяє перехоплювати дії та відправляти їх асинхронно або виконувати інші побічні ефекти. Загальним

вибором проміжного програмного забезпечення є Redux Thunk, який дозволяє обробляти асинхронні дії. «ConfigureStore» від RTK може легко включати middleware, таке як Redux Thunk, зменшуючи шаблонність і спрощуючи процес налаштування.

Redux вимагає написання функцій редюсера, які дотримуються певних принципів. Ці редуктори обробляють оновлення стану на основі відправлених дій. Хоча цей підхід забезпечує передбачуваність, він також може призвести до значної кількості шаблонного коду, особливо в міру того, як ваша програма зростає. RTK представляє концепцію редюсерів фрагментів, що змінює правила розробки Redux. Редюсери об’єднують три ключові аспекти:

Дії: вони визначають типи подій, які можуть відбуватися у вашій програмі, і дані, які вони містять. RTK надає допоміжні функції для узгодженого створення дій, що зменшує шаблонність.

Логіка редуктора: це ядро редуктора фрагмента, чиста функція, яка приймає поточний стан і дію як аргументи та повертає оновлений стан. Редуктори фрагментів RTK забезпечують структурований підхід до написання цієї логіки, сприяючи більш чистому та зручному для обслуговування коду.

Селектори: це функції, які витягують певні дані зі сховища Redux. RTK пропонує утиліти для створення мемоізованих селекторів, які можуть покращити продуктивність шляхом кешування результатів і уникнення непотрібних повторних обчислень.

Об’єднуючи ці елементи в єдині блоки, значно зменшується шаблонний код і сприяють кращій організації коду у вашій програмі Redux.

Крім спрощеного налаштування та скороченого шаблону, RTK пропонує додаткові переваги, які покращують як продуктивність програми, так і досвід розробника.

Оптимізація продуктивності: програми Redux можуть страждати від проблем з продуктивністю, якщо селектори, які отримують дані зі сховища, не оптимізовані. Функція «createSelector» RTK вирішує це, увімкнувши запам’ятовування. Запам'ятовування означає, що якщо фрагмент стану, на який покладається селектор, не змінився з моменту останнього виклику, результат селектора витягується з кешу замість того, щоб перераховуватися. Це може значно підвищити продуктивність, особливо для часто використовуваних селекторів.

Незмінні оновлення з помічниками незмінності: принципи Redux диктують, що редуктори мають бути чистими функціями, тобто вони не повинні змінювати існуючий об’єкт стану безпосередньо. Натомість вони повинні повернути новий об’єкт стану з необхідними змінами. Хоча це може призвести до більш детального коду, утиліта immer RTK пропонує рішення. Immer дозволяє писати логіку редуктора, яка, здається, змінює стан, але під капотом він перетворює ці оновлення на чисті функції. Це покращує читабельність і зручність обслуговування вашого коду редюсера, дотримуючись принципів Redux [8].

3.3.2 React Router DOM

React Router DOM є важливим інструментом для створення веб-застосунків на базі React. Цей додаток забезпечує потужні можливості навігації та маршрутизації, що є критично важливими для розробки сучасних односторінкових додатків (SPA), де змінюється вміст поточної сторінки без перезавантаження всієї сторінки.

React Router DOM включає компоненти, такі як Route, Link та Navigate, пропонуючи широкий набір інструментів для визначення маршрутів і забезпечення плавного переходу між ними.

Route - це основний компонент, який використовується для визначення шляхів. Цей компонент рендерить відповідний компонент, коли URL відповідає заданому маршруту.

Navigate - це сучасний інструмент для навігації, введений у версії 6 бібліотеки React Router DOM. Він використовується для програмного перенаправлення користувачів між сторінками, що дозволяє реалізувати складніші навігаційні шаблони

Компонент Link використовується для створення посилань у додатку, що забезпечує перехід між різними сторінками. Він змінює URL і викликає відповідний Route для відображення нового вмісту без перезавантаження сторінки.

Використання React Router DOM не тільки спрощує створення динамічних веб-застосунків з різним вмістом сторінок, але й покращує користувацький досвід, забезпечуючи швидку та зручну навігацію. Отже, без сумніву, React Router DOM є необхідним інструментом для будь-якого розробника React.

Розробники, які активно використовують цей інструмент, знають, що він дозволяє створювати потужну, добре організовану структуру навігації, яка направляє користувачів до необхідного контенту з мінімальними зусиллями. Це досягається завдяки гнучкості, яку надає бібліотека React Router DOM, дозволяючи розробникам ефективно керувати динамічними маршрутами у своїх застосунках.

Крім того, React Router DOM також включає компонент Routes, який допомагає ефективніше управляти маршрутами. Routes використовується для групування маршрутів, тобто компонентів Route.

Одним словом, React Router DOM є життєво важливим інструментом для розробників, які прагнуть створити React-додатки з динамічною і зручною навігацією. Завдяки йому, розробники можуть створювати високоякісні, зручні для користувачів додатки, які відповідають сучасним стандартам і вимогам веб-розробки [9].

3.3.3 React Hook Form

React Hook Form — популярна бібліотека, спеціально розроблена для спрощення обробки форм у програмах React. Забезпечує чистий та інтуїтивно зрозумілий API, побудований на основі хуків React.

React Hook Form дозволяє декларативно визначати логіку форми за допомогою таких хуків, як «useForm» і «useField». Цей підхід фокусується на тому, що повинна робити форма, а не загрузнути в деталях низькорівневої реалізації. Ви чітко та лаконічно описуєте поля форми, правила перевірки та поведінку подання.

Традиційне керування формами в React може включати багато шаблонного коду для обробки таких речей, як керування станом, перевірка форми та обробка помилок. React Hook Form усуває це, надаючи вбудовані механізми для цих функцій.

Бібліотека оптимізована для продуктивності. Він реалізує такі методи, як відкладена оцінка та запам’ятовування, щоб уникнути непотрібного повторного рендерингу та покращити загальну швидкість реагування ваших компонентів форми.

React Hook Form керує станом форми, включаючи значення кожного поля, їх дійсність і будь-які помилки, які можуть виникнути. Вам не потрібно вручну налаштовувати та керувати змінними стану для кожного елемента форми.

Бібліотека пропонує надійну систему перевірки, яка дозволяє визначати правила перевірки для кожного поля форми. Ви можете вказати правила перевірки безпосередньо у визначенні форми, роблячи логіку перевірки зрозумілою та легкою для обслуговування.

React Hook Form автоматично обробляє помилки під час надсилання форми або перевірки поля. Він надає доступ до повідомлень про помилки для кожного поля, дозволяючи відображати їх користувачеві та спрямовувати їх до виправлення будь-яких помилок.

Бібліотека чудово поєднується з іншими популярними бібліотеками в екосистемі React. Він добре інтегрується з React Router для обробки надсилання форм і навігації.

Декларативний підхід і вбудовані функції React Hook Form роблять розробку форм швидшою, легшою та менш схильною до помилок. Це дозволяє розробникам зосередитися на логіці своїх форм, а не загрузнути в деталях реалізації.

Відокремлюючи логіку форми від логіки компонентів, React Hook Form сприяє більш чистому та зручнішому коду. Ваші компоненти форми стають більш зосередженими та легшими для розуміння.

Загалом React Hook Form є чудовим вибором для розробників React, які хочуть спростити обробку форм у своїх програмах. Це спрощує процес, зменшує шаблонний код і сприяє зрозумілому коду, який зручно підтримувати. Якщо ви створюєте React-додатки, які включають форми, React Hook Form однозначно варто розглянути [10].

3.3.4 Emotion JS

CSS-in-JS — це підхід до стилізації, який інтегрує JavaScript із CSS для застосування стилів до веб-компонентів. Замість того, щоб писати стилі в окремих файлах CSS, бібліотеки CSS-in-JS дозволяють визначати стилі безпосередньо в коді JavaScript. Це дає кілька переваг:

CSS-код розміщено поряд із компонентом, який він стилізує, що полегшує розуміння того, як застосовуються стилі, і зменшує потребу перемикатися між файлами. Таке спільне розміщення сприяє кращій організації та спрощує технічне обслуговування, особливо для складних компонентів зі складним стилем.

CSS-in-JS дозволяє створювати динамічні стилі на основі пропів або стану компонента. Ви можете написати логіку в JavaScript, щоб умовно застосовувати

стилі або створювати стилі на основі введення користувача або інших динамічних умов. Це забезпечує більш чутливий та інтерактивний інтерфейс користувача.

Традиційний CSS іноді може призводити до конфліктів специфічності, коли стилі з різних частин вашої кодової бази неочікувано замінюють один одного. Бібліотеки CSS-in-JS зазвичай використовують механізми для створення унікальних імен класів, запобігаючи цим конфліктам і гарантуючи, що ваші стилі застосовуються за призначенням.

Багато бібліотек CSS-in-JS надають такі функції, як вихідні карти для стилів налагодження, мітки для зрозумілих людині назв класів і утиліти тестування для написання модульних тестів для стилів. Ці функції спрощують процес укладання та роблять його більш ефективним.

Хоча CSS-in-JS має переваги, важливо також враховувати його потенційні недоліки:

Загалом, CSS-in-JS — це потужна техніка, яка може покращити зручність обслуговування, гнучкість і досвід розробників у створенні стилів веб-застосунків. Якщо ви шукаєте спосіб створення більш модульних, динамічних і добре організованих інтерфейсів користувача, CSS-in-JS однозначно варто розглянути.

Emotion.js — популярна бібліотека CSS-in-JS, яка оптимізує стиль ваших веб-застосунків. На відміну від традиційного CSS, який зберігається в окремих файлах, Emotion дозволяє писати стилі CSS безпосередньо в коді JavaScript, пропонуючи кілька переваг:

Оптимізація продуктивності: Emotion надає параметри як для вбудованого стилю, так і для попередньої компіляції. Вбудований стиль підходить для розробки, оскільки забезпечує швидший час оновлення, тоді як попередня компіляція ідеальна для виробництва, оскільки вона генерує оптимізовані імена класів CSS, зменшуючи розмір пакетів і покращуючи час завантаження.

Передбачувані та безконфліктні стилі: Традиційний CSS іноді може призвести до проблем із специфічністю, коли стилі з різних частин вашої кодової бази неочікувано замінюють один одного. Emotion забезпечує чітку ієрархію для створення стилів, запобігаючи цим конфліктам і гарантуючи, що ваші стилі застосовуються за призначенням.

Покращений досвід розробника: Emotion містить функції, які роблять стиль більш приємним та ефективним. Карти вихідних кодів дозволяють налагоджувати стилі, вказуючи точний рядок коду, звідки виникає проблема. Мітки надають зрозумілі людині імена для згенерованих класів CSS, покращуючи читабельність. Утиліти тестування спрощують процес написання модульних тестів для ваших стилів, гарантуючи, що вони функціонують належним чином.

Emotion.js не залежить від фреймворків, тобто його можна легко інтегрувати з React, Vue або будь-яким іншим фреймворком JavaScript, який ви можете використовувати.

Emotion дозволяє створювати теми, які можна послідовно застосовувати у вашій програмі. Це особливо корисно для створення додатків із декількома темами, як-от світлий і темний режими, або для забезпечення узгодженої мови дизайну в усьому додатку.

Включивши Emotion.js у свій робочий процес розробки, ви зможете відчути переваги CSS-in-JS, створюючи чистіші, зручніші та продуктивніші веб-програми [11].

3.3.5 NodeJS та ExpressJS

Node.js і Express.js складають комбінацію технологій, які разом створюють потужну, гнучку і масштабовану платформу для розробки серверної частини веб-застосунків.

Node.js - це відкрите середовище для виконання коду JavaScript поза браузером. Використовуючи JavaScript-двигун V8, розроблений Google для браузера Chrome, Node.js забезпечує високу продуктивність і швидкість виконання коду. Він надає асинхронний, подіє-орієнтований підхід до виконання JavaScript, що дозволяє ефективно обробляти велику кількість одночасних запитів без блокування основного потоку виконання [12].

Express.js, у свою чергу, є веб-серверним фреймворком для Node.js. Його гнучкість і мінімалізм дозволяють розробникам отримувати максимальний контроль над процесом створення веб-додатків. Express.js пропонує широкий набір інструментів для обробки HTTP-запитів і формування відповідей, включаючи засоби для маршрутизації, шаблонізації, обробки помилок і багато іншого [13].

Використовуючи Node.js та Express.js разом, розробники можуть створювати веб-застосунки, які можуть обробляти великий обсяг даних і одночасно обслуговувати велику кількість користувачів. Вони можуть виконувати різні серверні завдання, такі як обробка HTTP-запитів з клієнтської частини, робота з базами даних, автентифікація користувачів, шифрування даних і багато іншого.

Крім того, як Node.js, так і Express.js мають великі спільноти розробників, які постійно працюють над удосконаленням цих технологій та надають підтримку іншим розробникам. Це робить їх привабливими для розробників різного рівня досвіду.

Таким чином, використання Node.js і Express.js для розробки серверної частини веб-застосунків дозволяє створювати високоякісні, швидкі, масштабовані та надійні веб-застоснуки, які відповідають високим стандартам сучасної веб-розробки.

3.3.6 Mongoose

Mongoose є незамінною бібліотекою для розробників Node.js, які використовують MongoDB як систему керування базами даних. Ця бібліотека дозволяє розробникам взаємодіяти з MongoDB на високому рівні абстракції, спрощуючи процес взаємодії з цією базою даних та підвищуючи ефективність розробки.

Mongoose відіграє важливу роль у тому, як розробники працюють з MongoDB. Вона забезпечує зручний API для роботи з даними, що дозволяє легко створювати, читати, оновлювати та видаляти записи в базі даних.

Однією з основних особливостей Mongoose є можливість визначення схем даних. Схеми дозволяють розробникам визначити структуру даних, які будуть зберігатися в MongoDB. Це може включати типи даних, обмеження, валідацію, значення за замовчуванням та інше.

Mongoose також пропонує багато корисних функцій, які полегшують роботу з MongoDB. Це включає різноманітні методи запитів, middleware для обробки запитів, вбудовані інструменти для валідації даних та навіть підтримку використання промісів замість зворотних викликів.

У контексті веб-застосунків, Mongoose використовується для створення моделей даних, які відображають структуру інформації, що має бути збережена в MongoDB. Це може включати все, від простих записів, таких як облікові записи користувачів, до складних структур даних, що охоплюють зв'язки між різними типами даних.

Крім того, Mongoose використовується для виконання всіх видів операцій з базою даних. Це може включати створення нових записів, оновлення наявних, читання записів з бази даних або видалення записів.

Загалом, Mongoose значно полегшує процес розробки веб-застосунків, які використовують MongoDB, дозволяючи розробникам зосередитися на створенні функціональності замість вирішення технічних питань взаємодії з базою даних [14].

4 РОЗРОБКА ВЕБ-ЗАСТОСУНКУ МЕНЕДЖЕР ЗАДАЧ

4.1 Архітектура додатку

Клієнтська сторона:

– відповідає за відображення інтерфейсу користувача та в залежності від того чи пройшла авторизація чи ні, має відображати основну сторінку додатку;

– перша сторінка це - вітальна сторінка;

– друга сторінка це форма реєстрації або логіну залежно від того чи є у користувача аккаунт;

– додаток включає дві основних сторінки які складаються з верхнього меню, бокового меню та робочої області;

– верхнє меню має функцію зміни кольорової схеми додатку на світлу та темну. Відображає ім’я користувача та його аватар. При натисканні на аватар відкривається вікно редагування даних користувача;

– бокове меню містить функції виходу з аккаунта, створення дошок, обрати фон та значки для дошки, редагування та видалення дошок, навігація між дошками та кнопку для переходу на сторінку працівників;

– сторінка працівників, відображає список всіх працівників, можна додавати, редагувати, видаляти працівників.

– робоча область відображає поточну обрану дошку. Тут міститься основний функціонал додатку. Можна створювати колонки, редагувати та видаляти їх. В кожній колонці можна створювати карточки з задачами. Карточка містить заголовок, опис, пріорітет виконання, крайню дату закінчення завдання, виконавця завдання та кнопки для редагування і видалення картки. Картки можна

перетягувати змінюючи їх положення в колонці або між колонками. Також можна фільтрувати карточки за міткою;

– виконує валідацію даних перед відправкою на сервер;

– взаємодіє з серверною стороною шляхом відправки запитів та відображає дані отримані від нього.

Серверна сторона:

– обробляє запити від клієнтської частини та надає відповіді про успішне виконання запиту або видає помилки які виникли при виконанні;

– взаємодіє з базою даних;

– перевіряє авторизацію користувача;

База даних:

– зберігає дані про користувача, дошки та працівників;

– виконує запити на отримання, додавання та зміну наявних даних у базі.

4.2 Реалізація функціональних можливостей

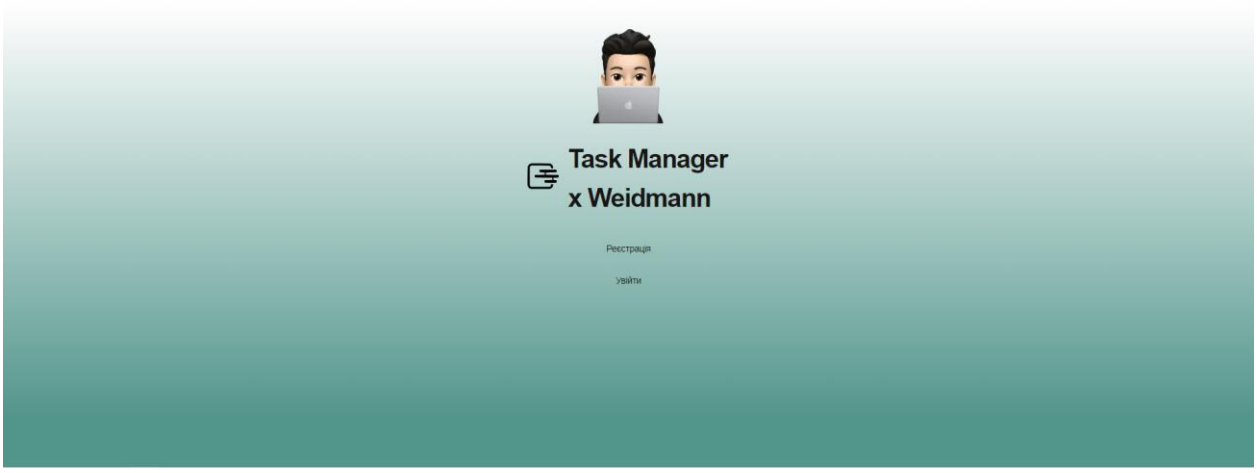


Рис. 4.1 Вітальна сторінка

При вході на сторінку застосунку користувач бачить привітальну сторінку, з якої можна перейти на сторінку реєстрації або логанізіції.

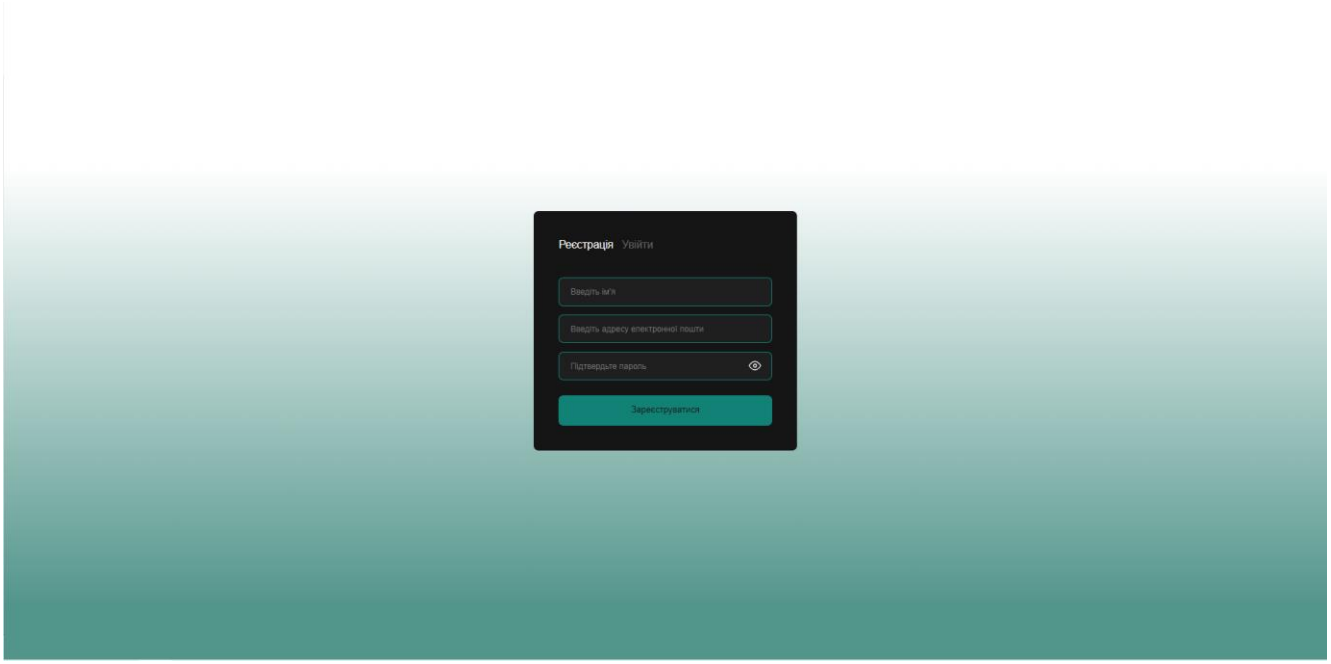


Рис. 4.2 Сторінка авторизації

При вході на сторінку реєстрації застосунку користувач бачить вікно реєстрації яке зображено на рисинку 4.2. Після заповнення форми на стороні клієнта відбувається валідація введених даних, а саме коректна пошта та мінімальна довжина паролю зображено на рисинку 4.2. Після успішною валідації дані відправляються на сервер де відбувається хешування паролю та створення користувача у базі даних. Після успішної реєстрації сервер повертає JWT токен який ми зберігаємо в локальне сховище і одразу виконується вхід в аккаунт.

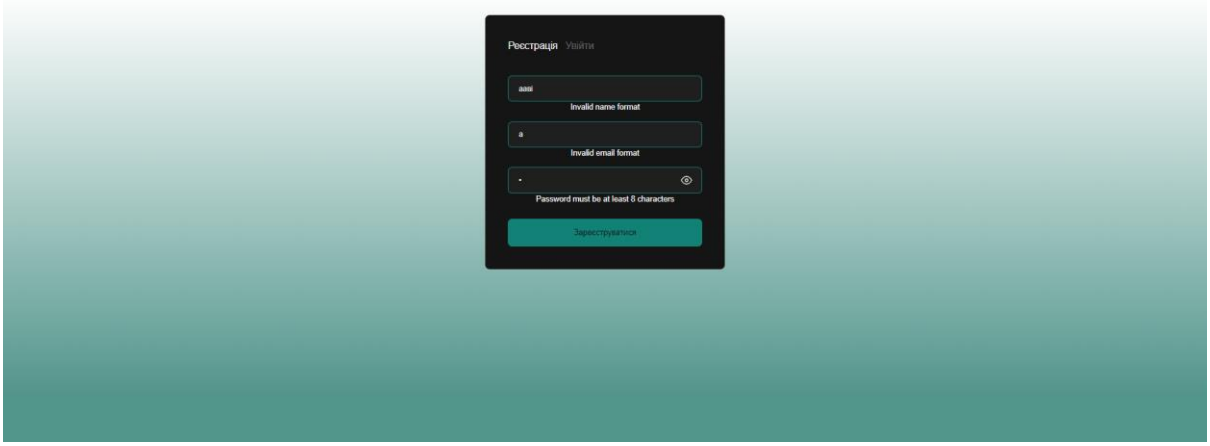


Рис. 4.3 Валідація данних

Авторизувавшись ми потрапляємо на головну сторінку рисунок 4.4 де бачимо бокове меню, верхнє меню і робочу область додатку які були описані вище, також можна побачити кнопку для переходу на сторінку працівників. Одразу після логіну додаток завантажує дошки користувача якщо вони існують. Якщо дошок нема в робочій області бачимо текст який підказує що треба створити дошку або можна натиснути на кнопку в боковому меню.

В верхньому меню ми можемо змінювати кольорову схему додатку на темну або світлу. Також ми бачимо наше ім’я та аватарку, яка також є кнопкою для відкриття модального вікна рисунок 4.5 де можна змінити ім’я, пошту та пароль користувача.

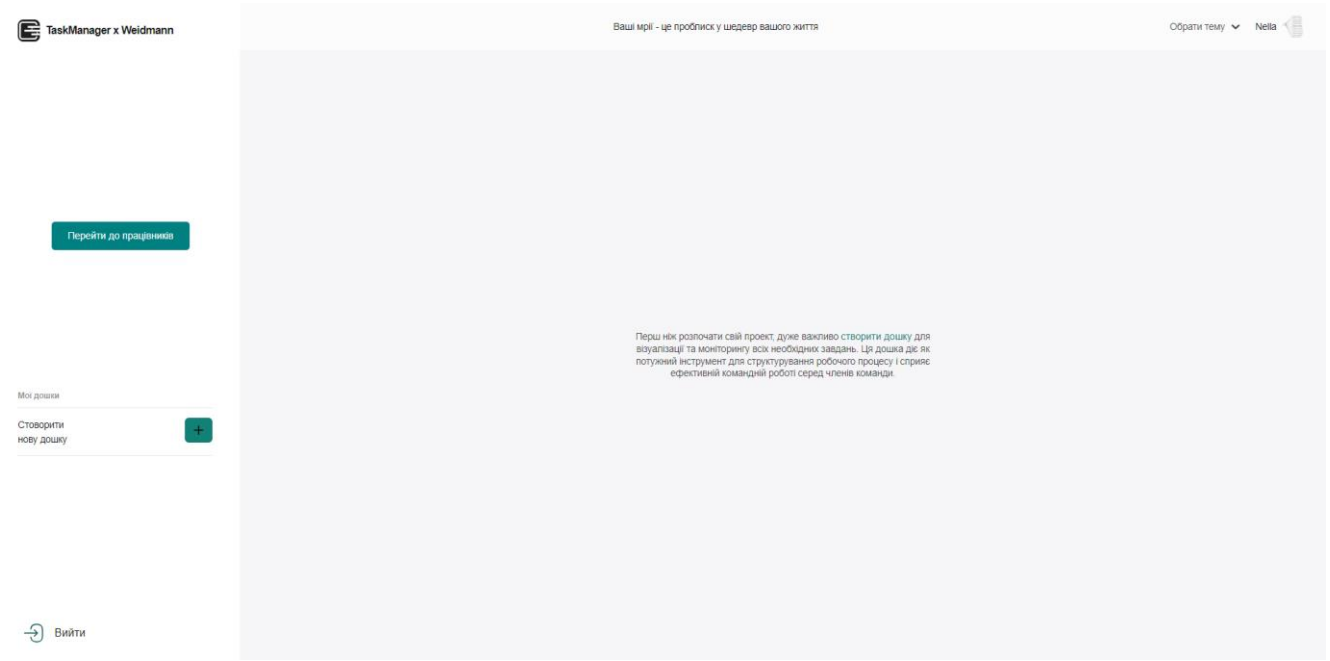


Рис. 4.4 Домашня сторінка застосунку

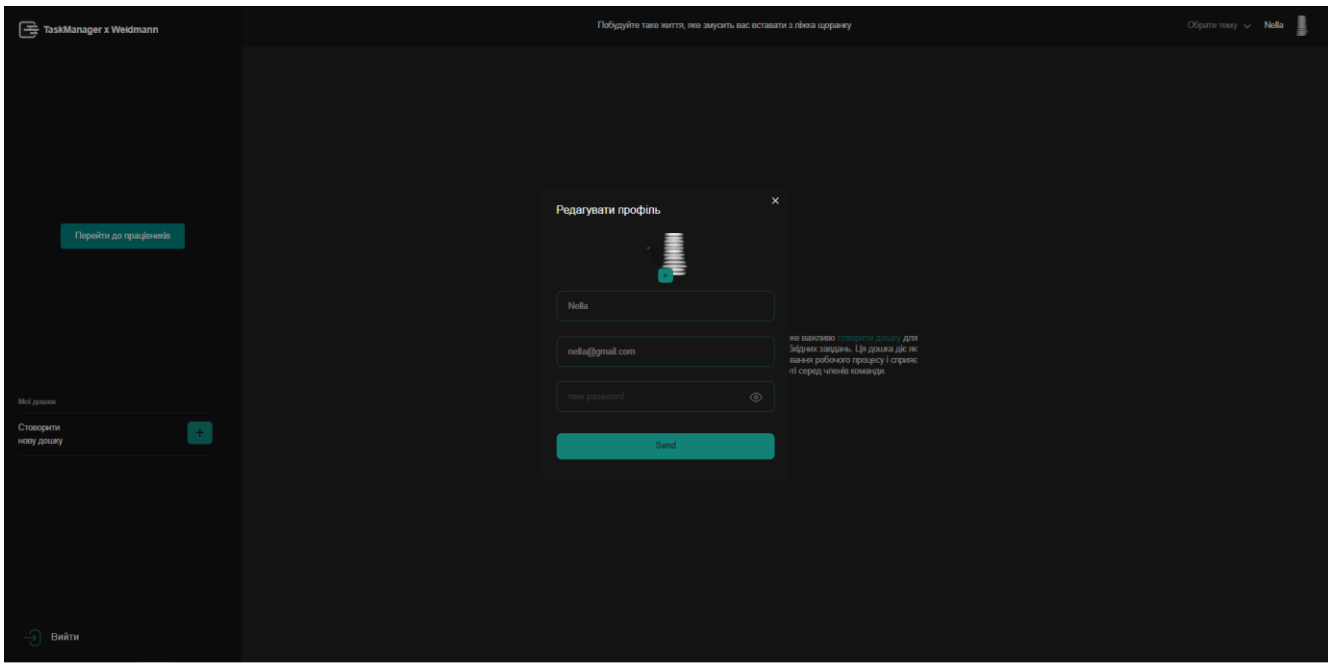


Рис. 4.5 Домашня сторінка застосунку

Для створення дошки натискаємо на кнопку створити дошку в боковому меню і відкриється модальна форма, рисунок 4.6, в якій ми обов’язково вказуємо назву дошки та можемо обрати фон іконки для карток. Після заповнення відправляється запит на сервер який додає дошку для користувача.

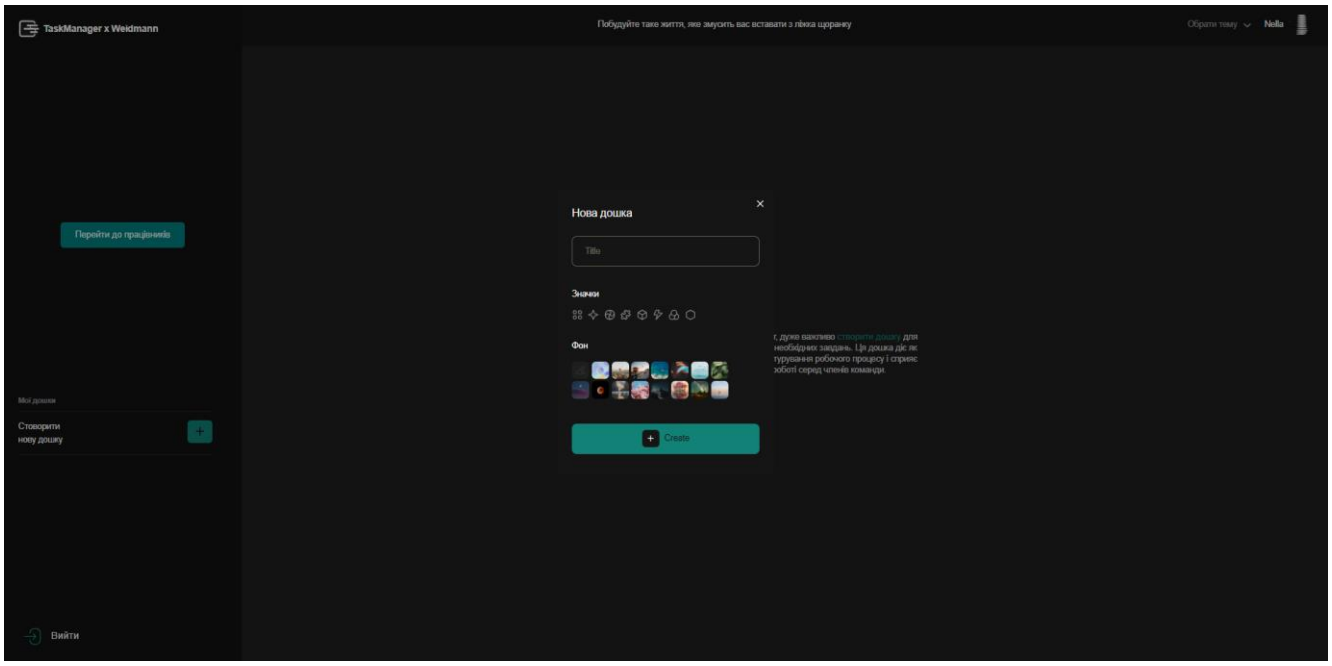


Рис. 4.6 Модальне вікно створення дошки

Створивши дошку одразу бачимо її в робочій області на рисунку 4.7. Тут ми маємо основний функціонал створюваного застосунку. Ми можемо створювати колонки і в колонках створювати картки.

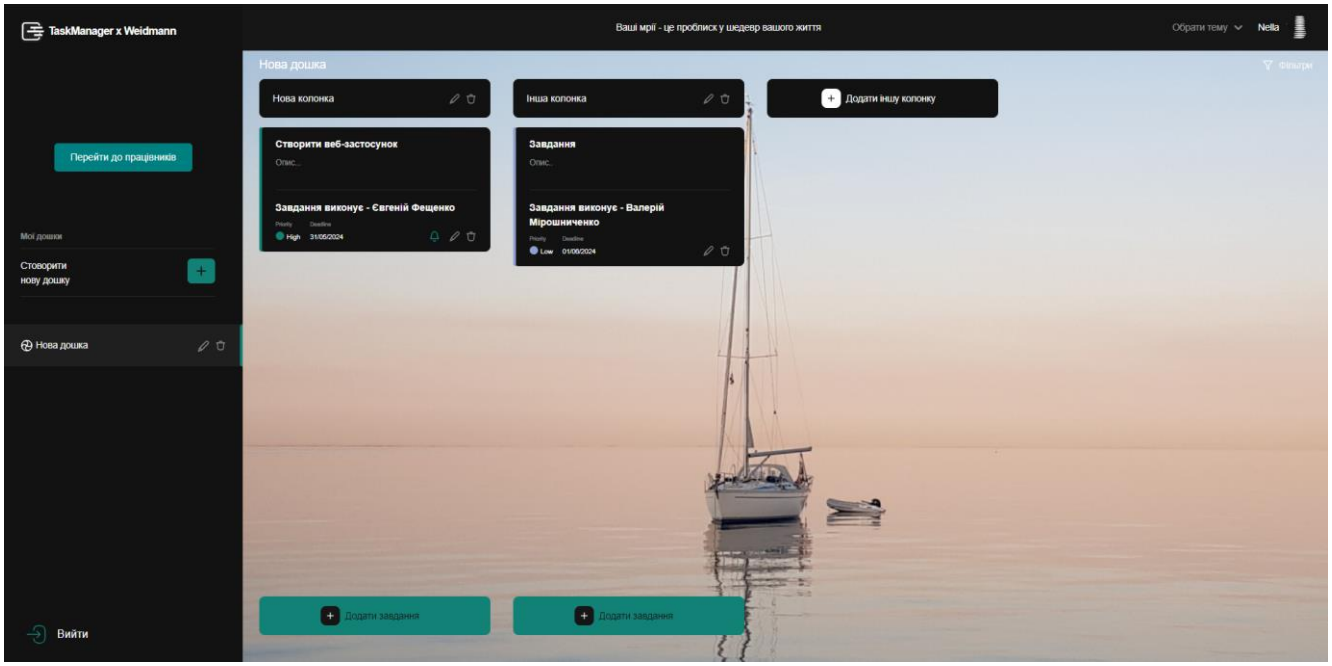


Рис. 4.7 Вигляд дошки з колонками та картками

Для колонок форма складається тільки з назви самої колонки. На рисунку 4.8 зображено форму яка складається з заголовку де визначається основна тема і опису де ми детальніше описуємо про що картка. Додатково можна додати мітку

для підкреслення важливості завдання та подальшого фільтрування карток і кінцеву дату виконання завдання також можна призначити хто буде виконувати завдання з працівників.

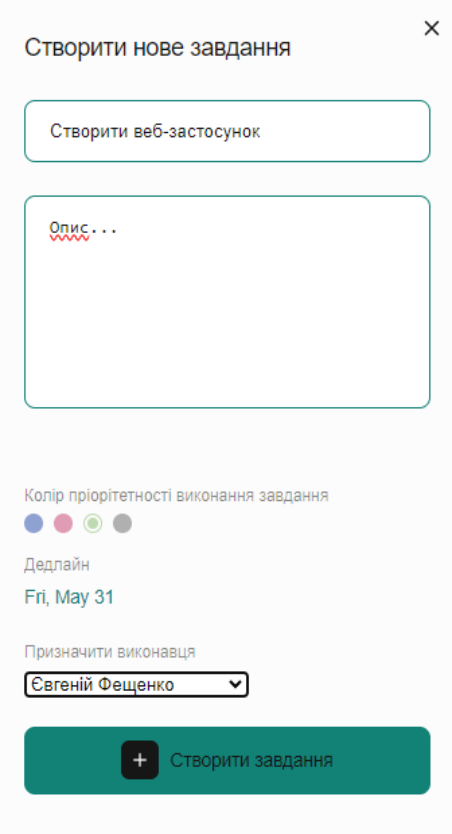


Рис. 4.8 Модальне вікно створення картки

Також важливою функцією є переміщення карток. На рисунку 4.9 зображено перетягування картки в іншу колонку або між картками поточної колонки.

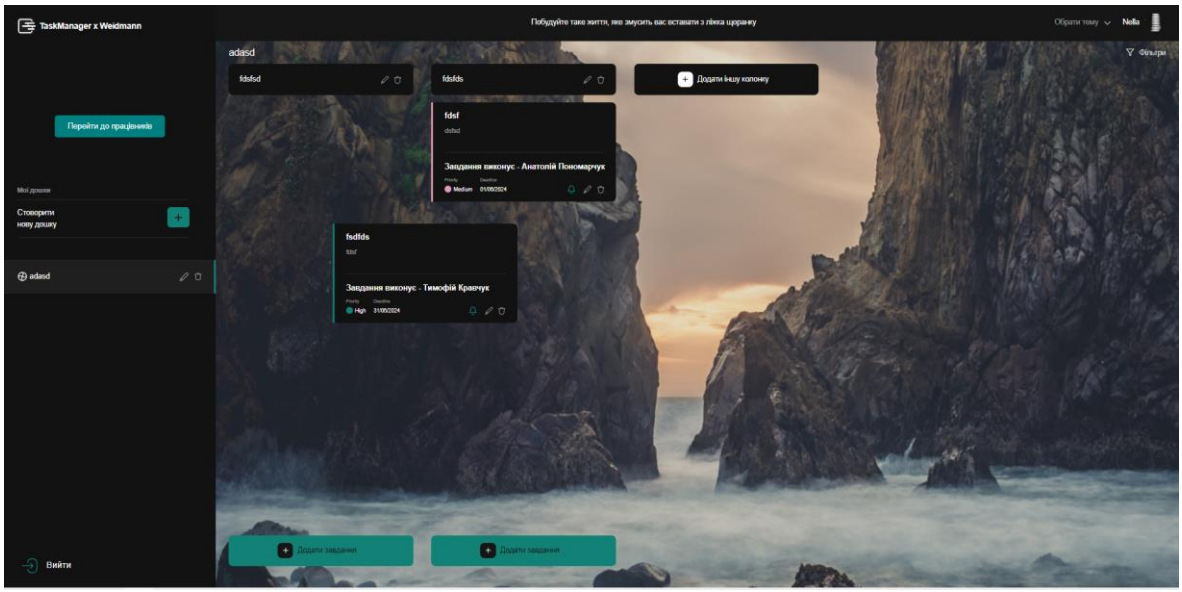


Рис. 4.9 Перетягування карток по дошці

Функція фільтрування карток по мітці яку ми вказуємо при створенні або можемо додати в редагуванні картки. Обравши фільтр ми будемо бачити лише картки з обраною міткою. І додатковою функцією є зміна фону дошки рисунок 4.10.

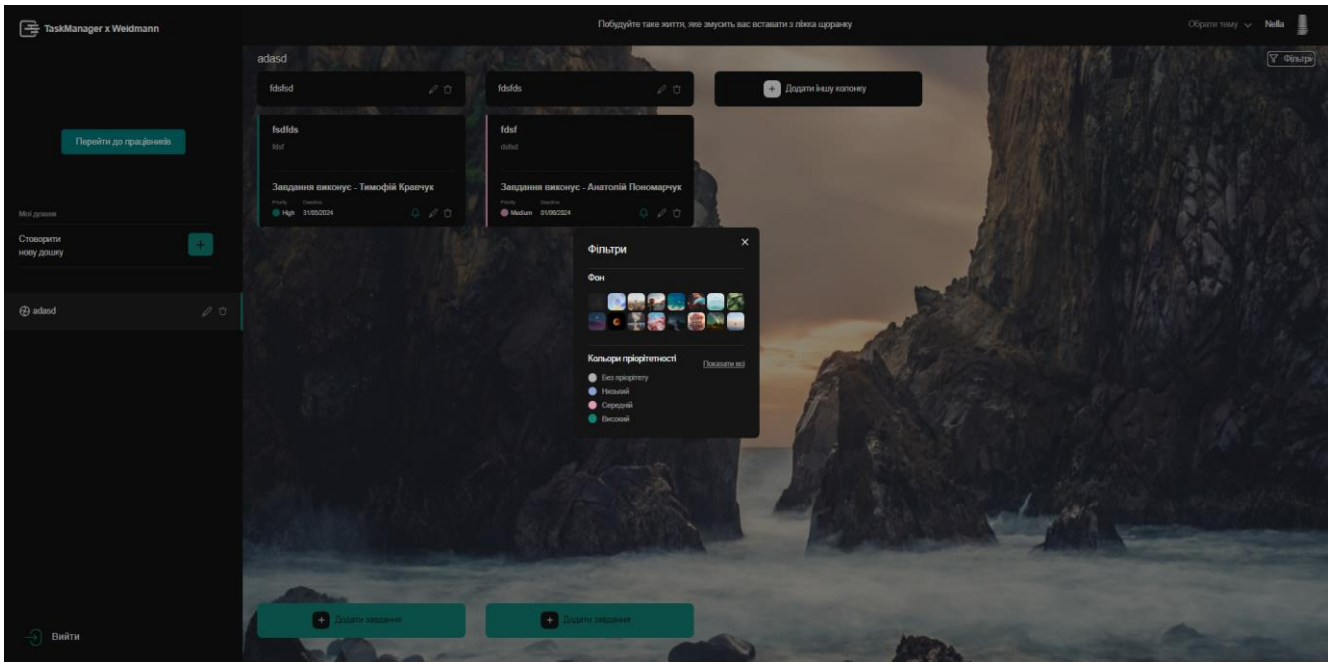


Рис. 4.10 Модальне вікно фільтрування карток

Також важливим функціоналом веб-застосунку є можливість роботи з базою даних працівників компанії «Вайдманн-МПФ». На рисунку 4.11 зображена сторінка працівників, де можна побачити список працівників, додати, редагувати, видалити працівника.

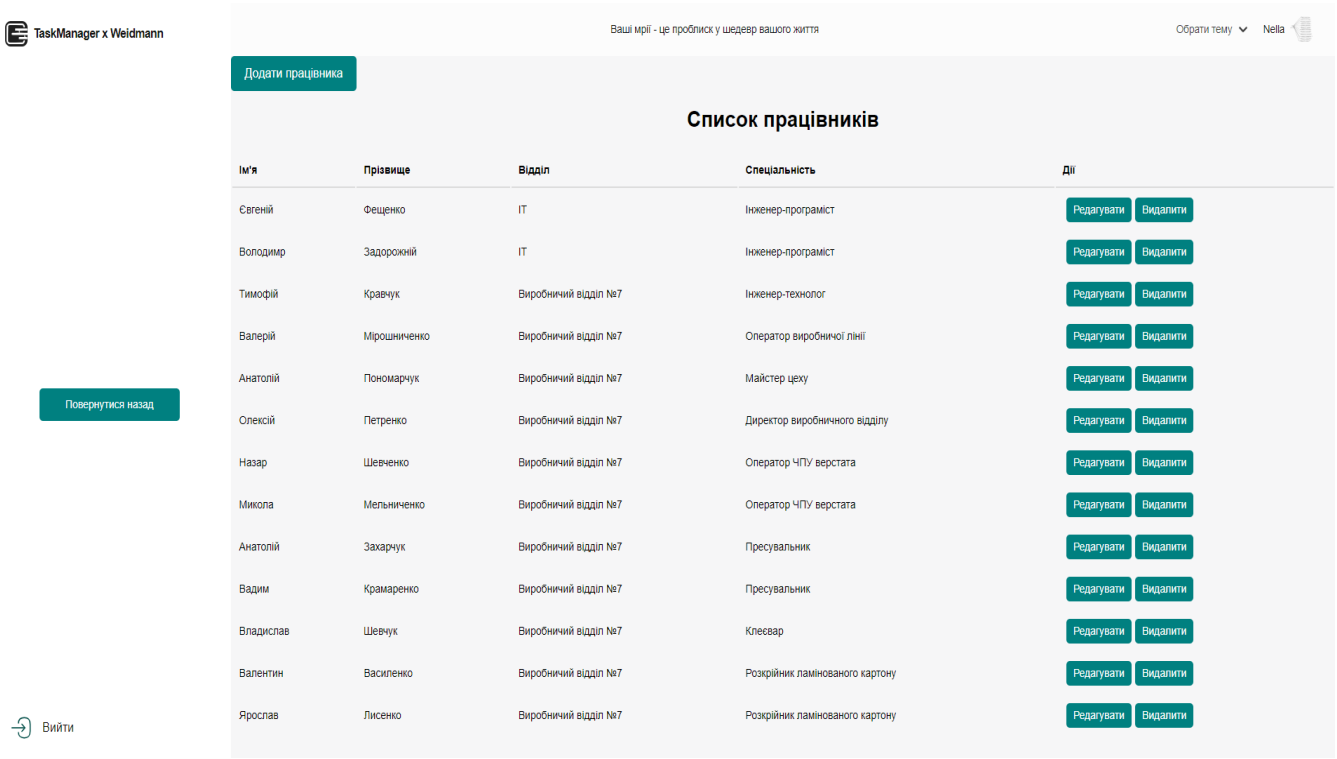


Рис. 4.11 Сторінка працівників

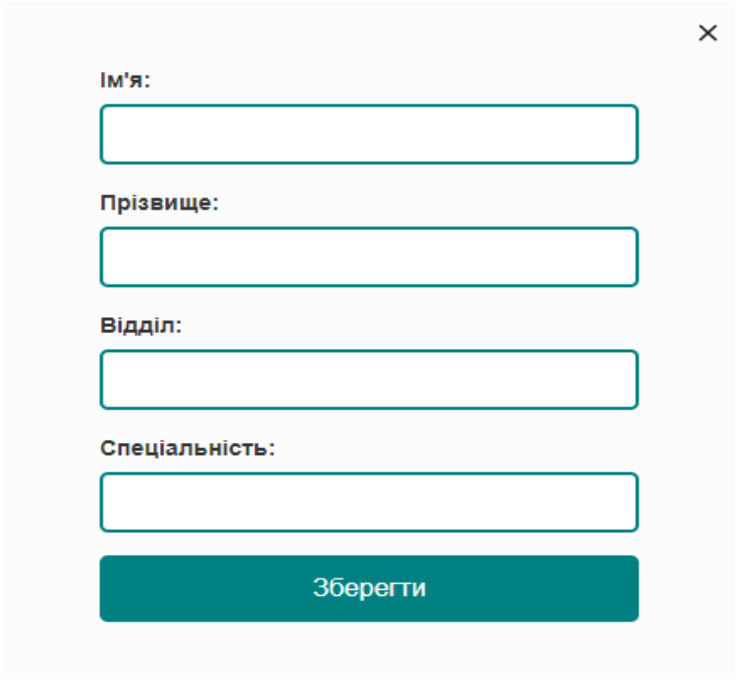


Рис. 4.12 Модальне вікно додавання нового працівника

ВИСНОВКИ

У цій дипломній роботі було успішно розроблено веб-застосунок менеджер задач. Цей веб-застосунок надає користувачам платформу для організації та управління своїми задачами, подібно до популярних інструментів, таких як Trello та Todist.

Розроблено та впроваджено зручний інтерфейс, який полегшує створення завдань, редагування, організацію за допомогою дошок і списків, а також функцію перетягування. Розроблено функціонал для робити з базою даних працівників компанії «Вайдманн-МПФ».

Розроблена логіка на стороні клієнта з використанням JavaScript та бібліотеки React для керування взаємодією користувачів і динамічного оновлення інтерфейсу користувача на основі дій користувача.

Створено модель даних для зберігання та керування інформацією про завдання в програмі. В базі даних ми маємо декілька моделей для кожного типу даних як користувач, дошка, колонки, картки, фон та працівники. В моделі користувача ми зберігаємо ім’я, пошту, хеш-пароль та токен. Модель дошки зберігає назву, посилання на картинку фону, іконку та айді користувача який створив дошку. Колонка містить лише назву та айді дошки. Модель картки містить всі дані з форми і айді колонки в якій вона знаходиться також містить айді працівника який виконує завдання. Модель фону містить айді та масив зображень для використання на дошках. Модель працівника містить айді, ім’я, прізвище, відділ, спеціальність.

Цей веб-застосунок менеджера завдань дозволяє підвищувати свою продуктивність і ефективно керувати своїми робочими навантаженнями.

Дана дипломна робота демонструє успішне створення функціонального веб-застосунку менеджера задач. Основні функціональні можливості забезпечують

міцну основу для подальшого розвитку та дослідження, прокладаючи шлях до більш комплексного рішення для управління задачами.

Робота пройшла апробацію:

Фещенко Є.В., Довженко Т.М. Розробка веб-додатку менеджер задач мовою JavaScript з використанням фреймворку React // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез. 24 квітня 2024 р., ДУІКТ, м. Київ – К: ДУІКТ, 2024. с.75.

Фещенко Є.В. Розробка веб-додатку планувальник завдань // «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)». Збірник тез. 20 травня 2024 р., ВНТУ, м. Вінниця – В: ВНТУ, 2024. Подано до друку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Скачков Д. А. Розробка практичних методів проектування та створення веб-додатків. ScienceRise. 2015. № 9(2). С. 22-26.
2. Веб-застосунок та його види [Електронний ресурс] – Режим доступу до ресурсу: https://webcase.com.ua/uk/blog/cho-takoe-web-prilozhenie-vse-vidy/.
3. Веб-застосунок Todoist [Електронний ресурс] – Режим доступу до ресурсу: https://todoist.com/?locale=en.
4. Веб-застосунок Trello [Електронний ресурс] – Режим доступу до ресурсу: https://trello.com/en.
5. Веб-застосунок Jira [Електронний ресурс] – Режим доступу до ресурсу: https://www.atlassian.com/software/jira.
6. Навчальний підручник JavaScript [Електронний ресурс] // JAVASCRIPT.INFO. – 2023. – Режим доступу до ресурсу: https://uk.javascript.info/.
7. React [Електронний ресурс] – Режим доступу: https://react.dev/
8. Redux Toolkit. [Електронний ресурс] – Режим доступу: https://redux-toolkit.js.org/
9. React Router DOM. [Електронний ресурс] – Режим доступу: https://reactrouter.com/en/main
10. React Hook Form. [Електронний ресурс] – Режим доступу: https://react-hook-form.com/
11. Emotion JS. [Електронний ресурс] – Режим доступу: https://emotion.sh/docs/introduction
12. NodeJs. [Електронний ресурс] – Режим доступу: https://nodejs.org/en
13. Express js. [Електронний ресурс] – Режим доступу: https://expressjs.com/
14. Mongoose. [Електронний ресурс] – Режим доступу: https://mongoosejs.com/
15. Banks A. Learning React: Functional Web Development with React and Redux / A. Banks, P. Eve. – Sebastopol: O\'Reilly Media, 2017. – 350 с.

16. Sammer B. Fullstack React: The Complete Guide to ReactJS and Friends. / B. Sammer, A. Bretz. – Sebastopol: Fullstack.io, 2017. – 835 с.
17. Casciaro M. Node.js Design Patterns / M. Casciaro, L. Mammino. – Birmingham: Packt Publishing, 2020. – 560 с.
18. Mardan A. Pro Express.js: Master Express.js: The Node.js Framework For Your Web Development. / Azat Mardan. – New York: Apress, 2014. – 330 с.
19. Wieruch R. The Road to React: Your journey to master plain yet pragmatic React.js / Robin Wieruch. – New York: Independently Published, 2020. – 238 с.
20. Hashimoto M. Vagrant: Up and Running / Mitchell Hashimoto. – Sebastopol: O\'Reilly Media, 2013. – 156 с.
21. Eoin K. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage 3rd Edition / K. Eoin, E. Brazil, S. Bradshaw. – Sebastopol: O\'Reilly, 2020. – 512 с.
22. Molinaro A. SQL Cookbook / Anthony Molinaro. – Sebastopol: O\'Reilly, 2005. – 500 с.
23. Flanagan D. JavaScript: The Definitive Guide: Master the World\'s Most-Used Programming Language / David Flanagan. – Sebastopol: O\'Reilly, 2020. – 706 с.
24. Simpson K. You Don\'t Know Js / Kyle Simpson. – Sebastopol: O\'Reilly, 2015. – 278 с.
25. Ackermann P. Full Stack Web Development: The Comprehensive Guide / Philip Ackermann. – Stuttgart: SAP Press, 2023. – 800 с.
26. Levy J. UX Strategy: How to Devise Innovative Digital Products that People Want 1st Edition / Jaime Levy. – Sebastopol: O\'Reilly, 2021. – 304 с.
27. Peterson C. Learning Responsive Web Design: A beginner\'s Guide / Clarissa Peterson. – Sebastopol: O\'Reilly, 2015. – 412 с.
28. Kapexhiu D. Building Microservices with Node.js: Explore microservices applications and migrate from a monolith architecture to microservices / Daniel Kapexhiu. – Birmingham: Packt Publishing, 2024. – 324 с.

29. Krause M. The Complete Developer: Master the Full Stack with TypeScript, React, Next.js, MongoDB, and Docker / Martin Krause. – San Francisco: No Starch Press, 2024. – 344 с.
30. Simpson J. How JavaScript Works: Master the Basics of JavaScript and Modern Web App Development / Jonathon Simpson. – New York: Apress, 2023. – 338 с.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка Web-застосунку менеджер задач мовою JavaScript з використанням фреймворку React

Виконав студент 4 курсу

Групи ПД-44
Фещенко Євгеній Вікторович
Керівник роботи

к.т.н., доцент кафдери ІПЗ Довженко Тимур Павлович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - спрощення процесу управління задачами за допомогою Web-застосунку створеного мовою JavaScript з використанням фреймворку React.
- **Об’єкт дослідження –** процес управління задачами.
- **Предмет дослідження –** Web-застосунок для управління задачами.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Огляд та аналіз існуючих Web-застосунків для управління задачами, визначити переваги та недоліки.
2. Визначити функціональні та нефункціональні вимого до Web-застосунку.
3. Проектування архітектури та структури Web-застосунку, враховуючи обрані технології та інструменти.
4. Розробка серверної та клієнтської частини Web-застосунку з використанням мови JavaScript та бібліотеки React.
5. Провести тестування Web-застосунку.

3

АНАЛІЗ АНАЛОГІВ

	Jira	Trello	Todoist	Task Manager
Створення, редагування, видалення дошок, колонок та карток	+	+	+	+
Можливість працювати з базою даних співробітників компанії “Weidmann”	-	-	-	+
Збереження даних на фізичному сервері компанії “Weidmann”	-	-	-	+
Нагадування про заплановану задачу	-	+	+	+

4

ВИМОГИ ДО ДОДАТКУ

Функціональні вимоги

1. Реєстрація та авторизація.
2. Створення та редагування дошок.
3. Створення і редагування колонок та карток на дошці.
4. Призначення завдань працівникам.
5. Перетягування карток по дошці.
6. Фільтрування та сортування карток за міткою.
7. Можливість вибрати тему веб-застосунку.
8. Редагування профіля користувача.
9. Створення, редагування, видалення працівників.

Нефункціональні вимоги

1. Адаптивний дизайн для коректного відображення на різних пристроях такі як: комп'ютер, ноутбук, планшет, телефон.
2. Забезпечення конфіденційності даних завдяки збереженню інформації на фізичному сервері компанії "Weidmann".

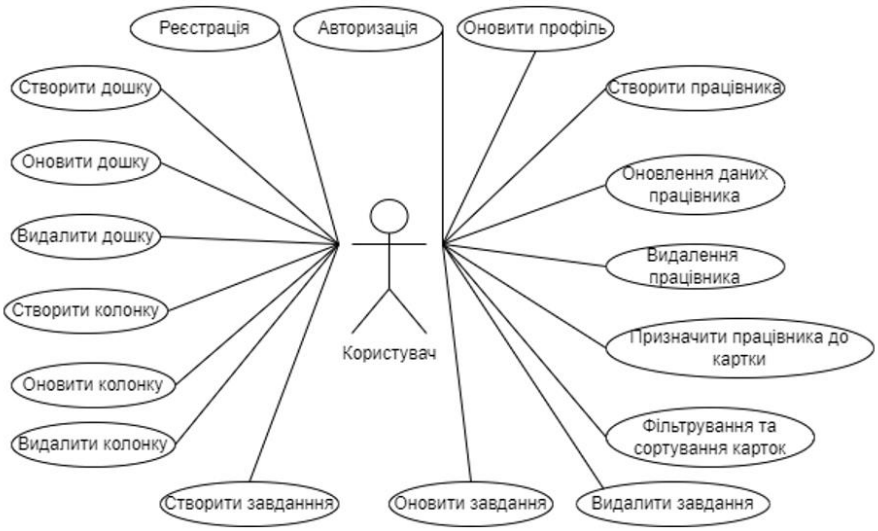
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Emotion

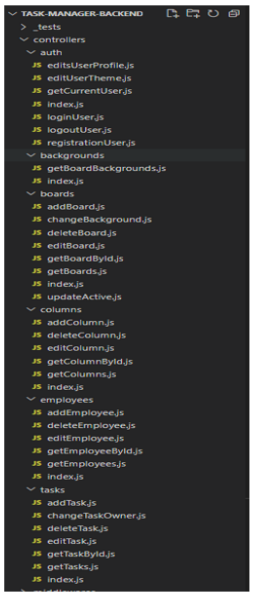
6

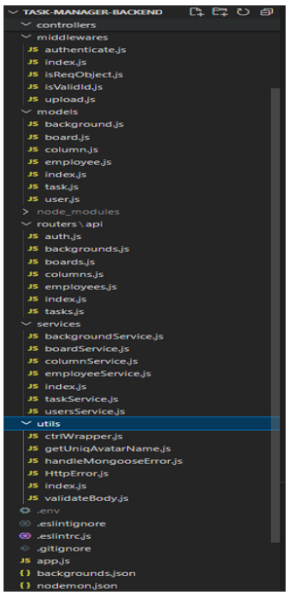
ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

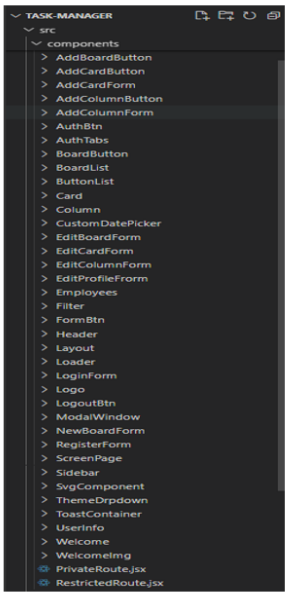


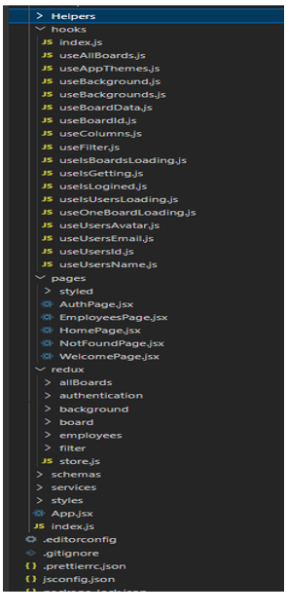
7

ФАЙЛОВА СТРУКТУРА ПРОЄКТУ









8

ДІАГРАМА КЛАСІВ

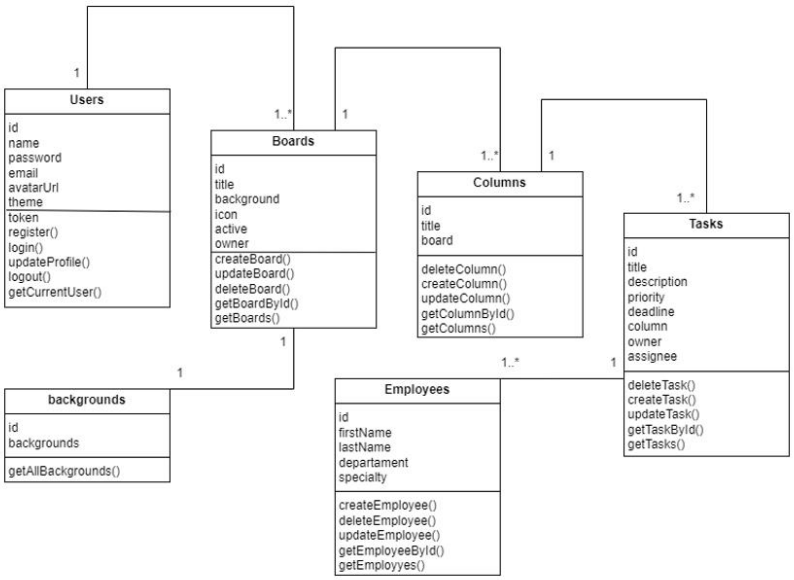
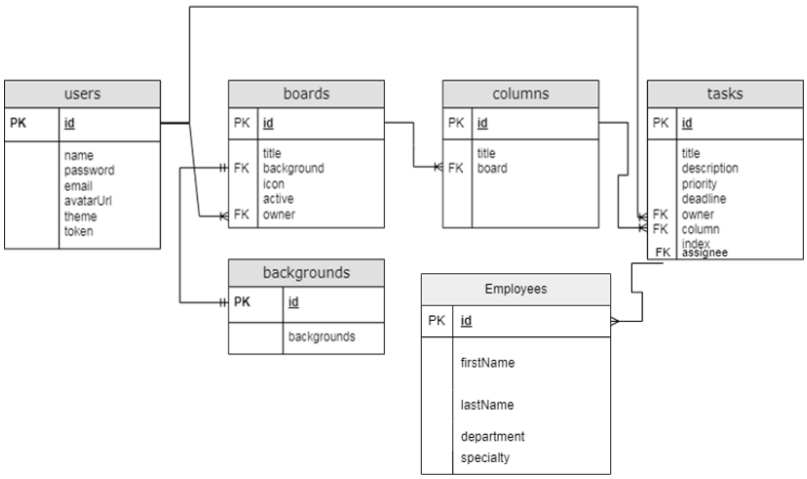


СХЕМА БАЗИ ДАНИХ



МАПА САЙТУ

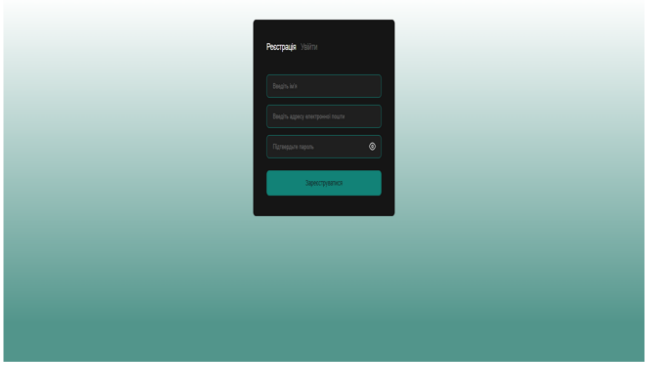
11

ЕКРАННІ ФОРМИ

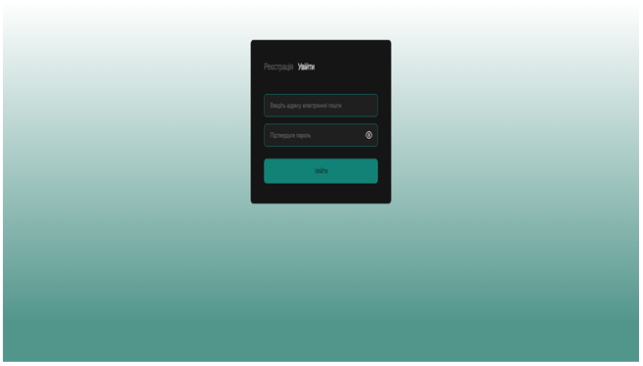
Вітальна сторінка

12

ЕКРАННІ ФОРМИ



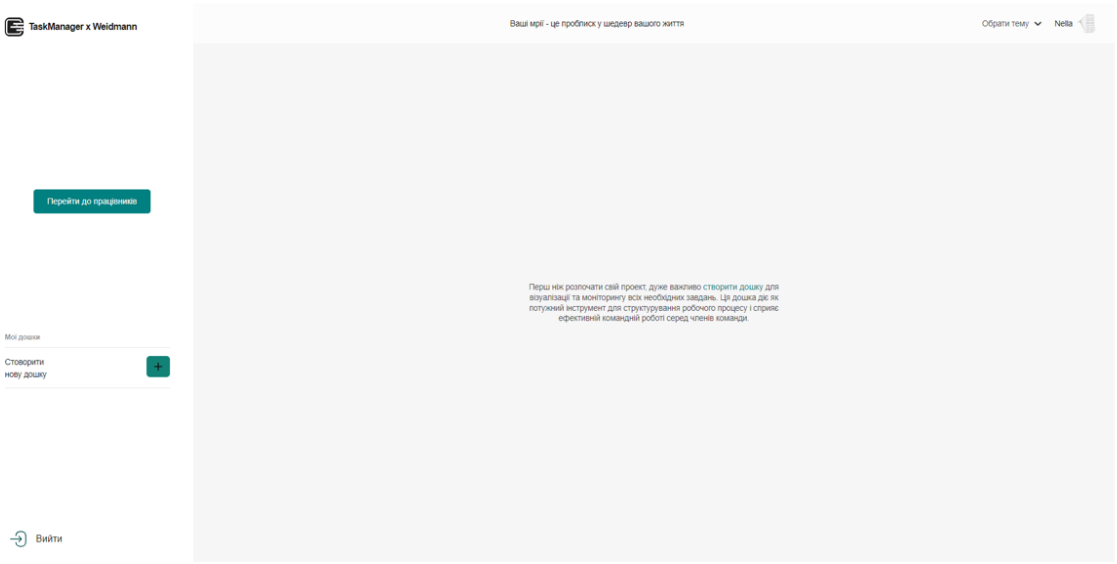
Сторінка реєстрації



Сторінка логанізації

13

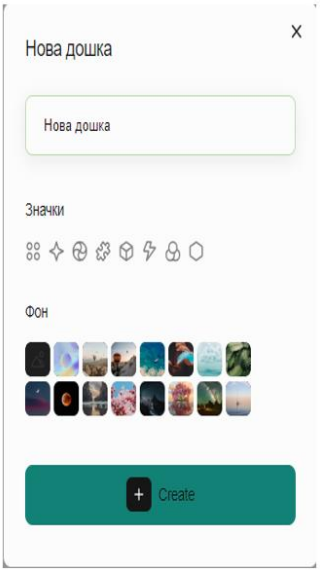
ЕКРАННІ ФОРМИ



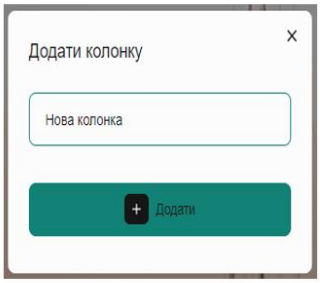
Домашня сторінка

14

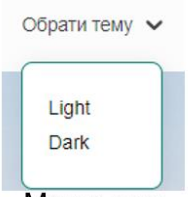
ЕКРАННІ ФОРМИ



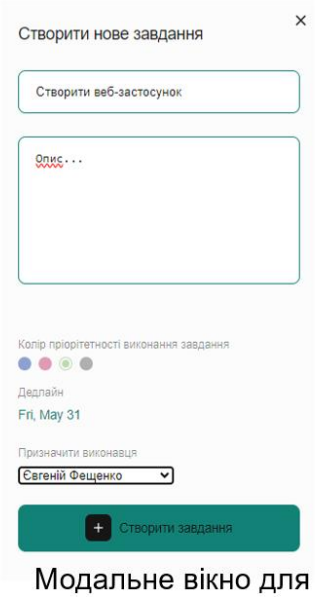
Модальне вікно для створення дошки



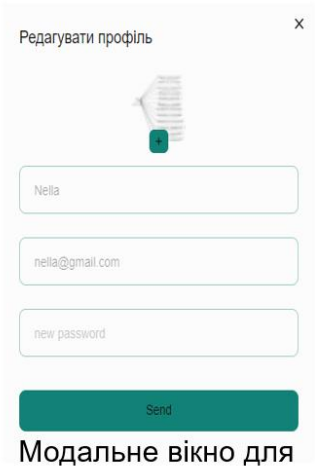
Модальне вікно для створення колонки



Меню для вибору теми

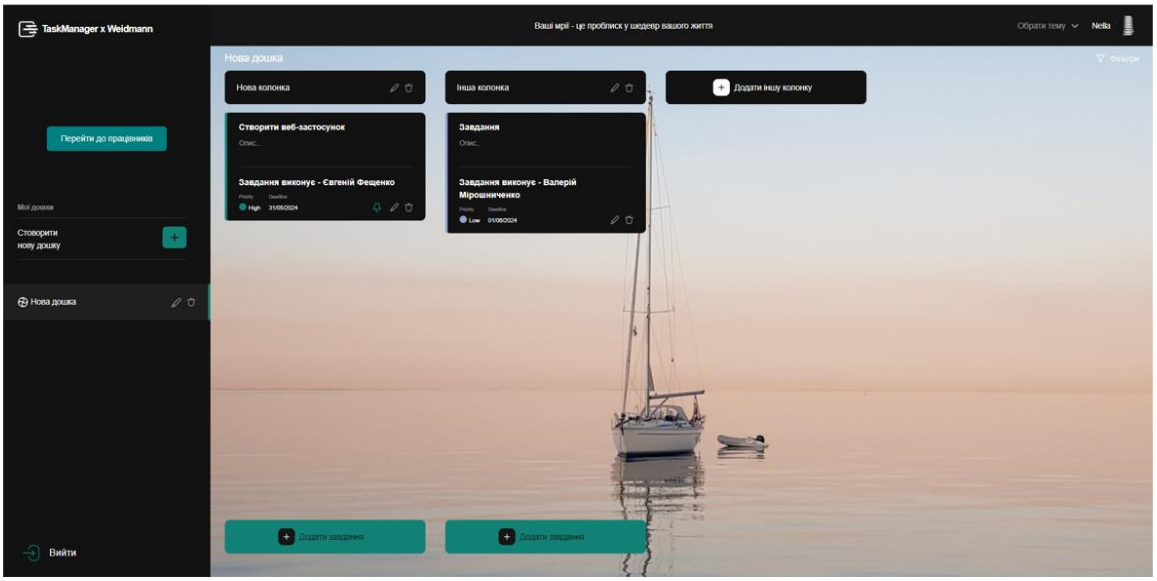


Модальне вікно для створення задачі



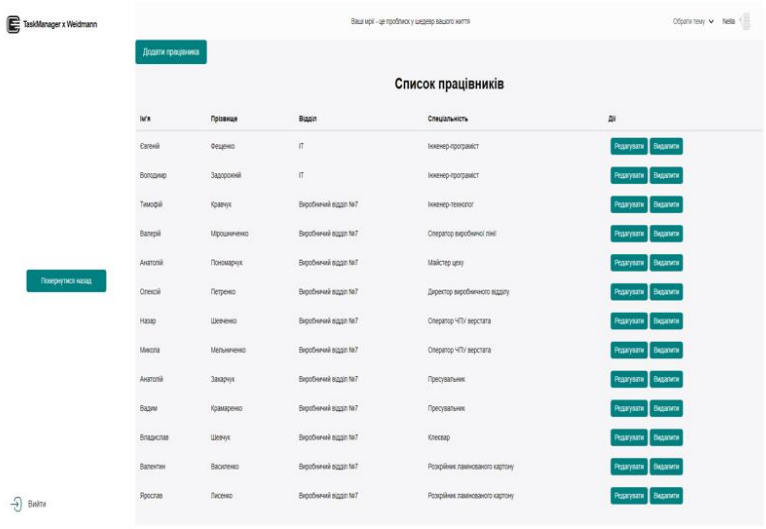
Модальне вікно для редагування профілю користувача

ЕКРАННІ ФОРМИ

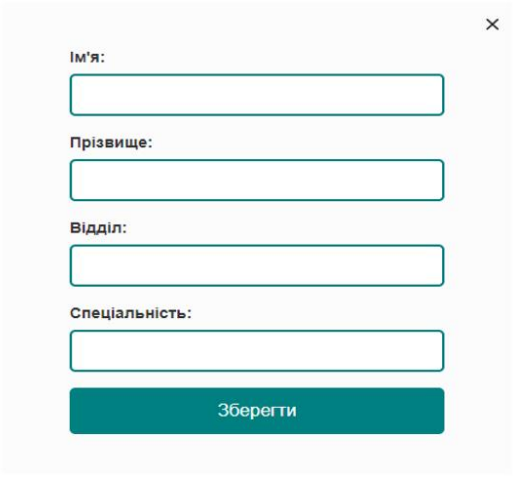


Вигляд дошки з колонками та картками

ЕКРАННІ ФОРМИ



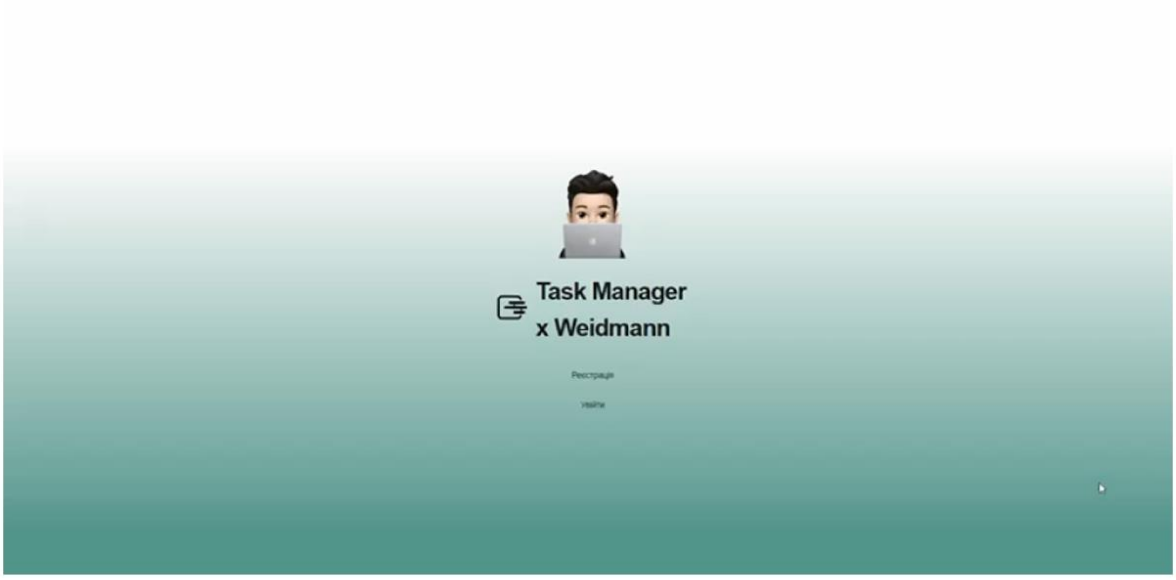
Сторінка списку працівників



Модальне вікно для додавання нового працівника

17

ВІДЕО РОБОТИ ВЕБ-ЗАСТОСУНКУ



18

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Фещенко Є.В. Розробка веб-додатку менеджер задач мовою JavaScript з використанням фреймворку React // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез. 24 квітня 2024 р., ДУІКТ, м. Київ – К: ДУІКТ, 2024. с.75.
2. Фещенко Є.В. РОЗРОБКА ВЕБ-ДОДАТКУ ПЛАНУВАЛЬНИК ЗАВДАНЬ // «МОЛОДЬ В НАУЦІ: ДОСЛІДЖЕННЯ, ПРОБЛЕМИ, ПЕРСПЕКТИВИ (МН-2024)». Збірник тез. 20 травня 2024 р., ВНТУ, м. Вінниця – В: ВНТУ, 2024. Подано до друку.

ВИСНОВКИ

1. Розглянуто та проаналізовано існуючі Web-застосунки для управління задачами, визначено переваги та недоліки.
2. Визначено функціональні та нефункціональні вимоги Web-застосунку.
3. Розроблено архітектуру та структуру Web-застосунку на базі технологій та інструментів React, Node.js, MongoDB.
4. Розроблено серверну та клієнтську частину Web-застосунку з використанням JavaScript та React.
5. Проведено тестування Web-застосунку.

ДОДАТОК Б ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
import axios from 'axios';
import { toast } from 'react-toastify';

const API_BASE_URL = 'https://task-manager-backend-
94yn.onrender.com';

const commonHeaders = {
  'Content-Type': 'application/json',
};

export const publicAxios = axios.create({
  baseURL: API_BASE_URL,
  headers: commonHeaders,
});

export const privateJsonAxios = axios.create({
  baseURL: API_BASE_URL,
  headers: commonHeaders,
});

export const privateFormDataAxios = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'multipart/form-data',
  },
});

privateJsonAxios.interceptors.request.use(
  async config => {
    const users = localStorage.getItem('persist:auth');
    const parsedUsers = JSON.parse(users);
    const tokens = parsedUsers.token.slice(1, -1);
    if (tokens) {
      if (config?.headers) {
        config.headers['Authorization'] = `Bearer
${tokens}`;
      }
    } else {
      return;
    }
    return config;
  },
  error => {
    return Promise.reject(error);
  }
);



privateFormDataAxios.interceptors.response.use(
  async response => {
    if ((response.data.status = 200)) {
      toast.success('Data were successfully updated');
      return response;
    }
    return response;
  },
  async error => {
    if (error.response.status === 500) {
      toast.error('Something went wrong. Please try again
later.');
    }
    if (error.response.status === 401) {
      return (window.location.href = '/task-
manager/authentication/login');
    }
    if (error.response.status === 400) {
      toast.error('Something went wrong. Please report us
an error!');
    }
    if (error.response.status === 409) {
      toast.error(
        'User with such email is already exist, please enter
another email'
      );
    }
    return Promise.reject(error);
  }
);

privateJsonAxios.interceptors.response.use(
  async response => {
    if (response.data.code === 201 &&
response.config.url === '/api/boards') {
      toast.success('New board successfully added');
      return response;
    }
    if (response.data.code === 200 &&
response.config.url === '/api/email') {
      toast.success('Your question sent. We answer your as
soon as posiible.');
      return response;
    }

    if (response.data.code === 201 &&
response.config.url === '/api/columns') {
      toast.success('New column successfully added');
      return response;
    }

    if (response.data.code === 201 &&
response.config.url === '/api/tasks') {
      toast.success('New card successfully added');
      return response;
```

```
    }
    return response;
  },
  async error => {
    if (error.response.status === 500) {
      toast.error('Something went wrong. Please try again
later.');
    }
    if (error.response.status === 401) {
      return (window.location.href = '/task-
manager/authentication/login');
    }
    if (error.response.status === 400) {
      toast.error('Something went wrong. Please report the
error to us!');
    }
    if (error.response.status === 409 && error.config.url
=== '/api/boards') {
      toast.error(
        'A board with that name already exists, please enter
another name'
      );
    }
    return Promise.reject(error);
  }
);

import { configureStore } from '@reduxjs/toolkit';
import { setupListeners } from '@reduxjs/toolkit/query';
import {
  persistStore,
  persistReducer,
  FLUSH,
  REHYDRATE,
  PAUSE,
  PERSIST,
  PURGE,
  REGISTER,
} from 'redux-persist';
import storage from 'redux-persist/lib/storage';

import { authenticationReducer } from
'./authentication/slice';
import { allBoardsReducer } from './allBoards/slice';
import { singleBoardReducer } from './board/slice';
import { backgroundsDataReducer } from
'./background/slice';
import { priorityFilterReducer } from './filter/slice';

const authenticationPersistConfig = {
  key: 'auth',
  storage,
  whitelist: ['token', 'users', 'theme'],
};

const boardsPersistConfig = {
  key: 'boards',
  storage,
};

const singleBoardPersistConfig = {
  key: 'board',
  storage,
};

const allBackgroundsConfig = {
  key: 'backgrounds',
  storage,
};

const priortyFilterPersistConfig = {
  key: 'priorityFilter',
  storage,
};

export const store = configureStore({
  reducer: {
    auth: persistReducer(authenticationPersistConfig,
authenticationReducer),
    boards: persistReducer(boardsPersistConfig,
allBoardsReducer),
    board: persistReducer(singleBoardPersistConfig,
singleBoardReducer),
    backgrounds: persistReducer(allBackgroundsConfig,
backgroundsDataReducer),
    filter: persistReducer(priortyFilterPersistConfig,
priorityFilterReducer),
  },

  middleware: getDefaultMiddleware =>
    getDefaultMiddleware({
      serializableCheck: {
        ignoredActions: [FLUSH, REHYDRATE, PAUSE,
PERSIST, PURGE, REGISTER],
      },
    }),

  devTools: process.env.NODE_ENV === 'development',
});

setupListeners(store.dispatch);

export const persistor = persistStore(store);

import { privateJsonAxios } from 'services/axios';
import { createAsyncThunk } from '@reduxjs/toolkit';

export const fetchAllBoards = createAsyncThunk(
  'boards/fetchAll',
  async (_, thunkAPI) => {
    try {
      const { data } = await
privateJsonAxios.get('/api/boards');
      return data;
    } catch (error) {
      return thunkAPI.rejectWithValue(error.code);
```

```
    }
  }
);

export const addNewBoards = createAsyncThunk(
  'boards/addNew',
  async (board, thunkAPI) => {
    try {
      const { data } = await
privateJsonAxios.post('/api/boards', board);
      return data;
    } catch (error) {
      return thunkAPI.rejectWithValue(error.code);
    }
  }
);

export const updatesBoardStatus = createAsyncThunk(
  'boards/updateStatus',
  async (boardData, thunkAPI) => {
    try {
      const { data } = await privateJsonAxios.patch(
        `api/boards/${boardData.boardId}/active`,
        boardData.body
      );
      return data;
    } catch (error) {
      return thunkAPI.rejectWithValue(error.code);
    }
  }
);

export const editBoardsById = createAsyncThunk(
  'boards/editById',
  async (boardData, thunkAPI) => {
    try {
      const { data } = await privateJsonAxios.put(
        `/api/boards/${boardData.boardId}`,
        boardData.body
      );
      return data;
    } catch (error) {
      return thunkAPI.rejectWithValue(error.code);
    }
  }
);

export const deleteBoards = createAsyncThunk(
  'boards/delete',
  async (boardId, thunkAPI) => {
    try {
      const { data } = await
privateJsonAxios.delete(`/api/boards/${boardId}`);
      return data;
    } catch (error) {
      return thunkAPI.rejectWithValue(error.message);
    }
  }
);

export const editBoardsBackground =
createAsyncThunk(
  'boards/editBackground',
  async (boardData, thunkAPI) => {
    try {
      const { data } = await privateJsonAxios.patch(
        `/api/boards/${boardData.id}/background`,
        boardData.body
      );
      return data;
    } catch (error) {
      return thunkAPI.rejectWithValue(error.message);
    }
  }
);

import { createSlice } from '@reduxjs/toolkit';
import {
  fetchAllBoards,
  addNewBoards,
  updatesBoardStatus,
  editBoardsById,
  deleteBoards,
  editBoardsBackground,
} from './operations';

const initialState = {
  allBoardInfo: [],
  isLoading: false,
};

export const allBoardsSlice = createSlice({
  name: 'boards',
  initialState,
  reducers: {},
  extraReducers: builder =>
    builder
      .addCase(fetchAllBoards.fulfilled, (state, action) =>
{
        state.allBoardInfo = action.payload.data.boards;
        state.isLoading = false;
      })
      .addCase(fetchAllBoards.pending, state => {
        state.isLoading = true;
      })
      .addCase(fetchAllBoards.rejected, state => {
        state.isLoading = false;
      })
      .addCase(addNewBoards.fulfilled, (state, action) =>
{
        state.allBoardInfo.push(action.payload.data.board);
        state.isLoading = false;
      })
      .addCase(addNewBoards.pending, state => {
        state.isLoading = true;
      })
```

```
    .addCase(addNewBoards.rejected, state => {
      state.isLoading = false;
    })
    .addCase(updatesBoardStatus.fulfilled, (state, action)
=> {
      const boardsIdToUpdate =
action.payload.data.board._id;
      const changeBoard = state.allBoardInfo.find(
        board => board._id === boardsIdToUpdate
      );
      if (changeBoard) {
        changeBoard.active =
action.payload.data.board.active;
      }
    })

    .addCase(editBoardsById.fulfilled, (state, action) =>
{
      const { _id: boardId } = action.payload.data.board;
      const indexes = state.allBoardInfo.findIndex(
        board => board._id === boardId
      );

      state.allBoardInfo.splice(indexes, 1,
action.payload.data.board);
    })
    .addCase(editBoardsById.pending, state => {
      state.isLoading = true;
    })
    .addCase(editBoardsById.rejected, state => {
      state.isLoading = false;
    })
    .addCase(deleteBoards.fulfilled, (state, action) => {
      const deletedBoardById =
action.payload.data.deletedBoard._id;
      state.allBoardInfo = state.allBoardInfo.filter(
        board => board._id !== deletedBoardById
      );
      state.isLoading = false;
    })
    .addCase(deleteBoards.pending, state => {
      state.isLoading = true;
    })
    .addCase(deleteBoards.rejected, state => {
      state.isLoading = false;
    })
    .addCase(editBoardsBackground.pending, state => {
      state.isLoading = true;
    })
    .addCase(
      editBoardsBackground.fulfilled,
      (
        state,
        {
          payload: {
            data: { board },
          },
        }
      ) => {
        const boardsId = board._id;
        state.isLoading = false;
        state.allBoardInfo = [
          ...state.allBoardInfo.map(board =>
            board._id === boardsId
              ? { ...board, background: board.background }
              : board
          ),
        ];
      }
    )

    .addCase(editBoardsBackground.rejected, state => {
      state.isLoading = false;
    }),
});

export const allBoardsReducer = allBoardsSlice.reducer;

import { createAsyncThunk } from '@reduxjs/toolkit';

import {
  publicAxios,
  privateJsonAxios,
  privateFormDataAxios,
} from 'services/axios';

export const registerUser = createAsyncThunk(
  'auth/register',
  async (user, thunkAPI) => {
    try {
      const { data } = await
publicAxios.post(`/api/users/register`, user);
      const { email, password } = user;
      let userData;
      if (data.status === 'success') {
        const { data } = await
publicAxios.post('/api/users/login', {
          email,
          password,
        });
        userData = data;

        return userData;
      }
      return userData;
    } catch (error) {
      return thunkAPI.rejectWithValue(error.code);
    }
  }
);

export const loginUser = createAsyncThunk(
  'auth/login',
  async (user, thunkAPI) => {
    try {
```

```
    const { data } = await
publicAxios.post(`/api/users/login`, user);
    return data;
   } catch (error) {
    return thunkAPI.rejectWithValue(error.code);
   }
  }
);

export const logOutUser = createAsyncThunk(
  'auth/logout',
  async (_, thunkAPI) => {
   try {
    await privateJsonAxios.post('/api/users/logout');

    return;
   } catch (error) {
    return thunkAPI.rejectWithValue(error.code);
   }
  }
);

export const getCurrentUser = createAsyncThunk(
  'auth/current',
  async (_, thunkAPI) => {
   try {
    const { data } = await
privateJsonAxios.get(`/api/users/current`);
    if (!data) {
     throw new Error();
    }
    return data;
   } catch (error) {
    return thunkAPI.rejectWithValue(error.code);
   }
  }
);

export const editUserProfile = createAsyncThunk(
  'auth/profile',
  async (userData, thunkAPI) => {
   try {
    const { data } = await privateFormDataAxios.put(
     `/api/users/current/${userData.userId}`,
     userData.formData
    );
    return data;
   } catch (error) {
    return thunkAPI.rejectWithValue(error.code);
   }
  }
);

export const editUserTheme = createAsyncThunk(
  'auth/theme',
  async (userData, thunkAPI) => {
   try {
    const { data } = await privateJsonAxios.patch(
     `/api/users/current/${userData.id}/theme`,
     userData.body
    );
    return data;
   } catch (error) {
    return thunkAPI.rejectWithValue(error.code);
   }
  }
);

import { createSlice } from '@reduxjs/toolkit';
import {
  getCurrentUser,
  registerUser,
  loginUser,
  logOutUser,
  editUserTheme,
  editUserProfile,
} from './operations';

const authInitialState = {
  users: { name: '', email: '', avatar: '', id: '' },
  token: '',
  theme: 'dark',
  isLogined: false,
  isLoading: false,
  isGetCurrentUser: false,
};
const authenticationSlice = createSlice({
  name: 'auth',
  initialState: authInitialState,
  extraReducers: builder =>
   builder
    .addCase(registerUser.fulfilled, (state, action) => {
     state.users.name = action.payload.data.user.name;
     state.users.email = action.payload.data.user.email;
     state.users.avatar =
action.payload.data.user.avatarUrl;
     state.users.id = action.payload.data.user._id;
     state.token = action.payload.data.token;
     state.theme = action.payload.data.user.theme;
     state.isLogined = true;
     state.isLoading = false;
    })
    .addCase(registerUser.pending, state => {
     state.isLoading = true;
    })
    .addCase(registerUser.rejected, state => {
     state.isLoading = false;
    })
    .addCase(loginUser.fulfilled, (state, action) => {
     state.users.name = action.payload.data.user.name;
     state.users.email = action.payload.data.user.email;
     state.users.avatar =
action.payload.data.user.avatarUrl;
     state.users.id = action.payload.data.user._id;
     state.token = action.payload.data.token;
     state.theme = action.payload.data.user.theme;
```

```
        state.isLogined = true;
        state.isLoading = false;
      })
      .addCase(loginUser.pending, state => {
        state.isLoading = true;
      })
      .addCase(loginUser.rejected, state => {
        state.isLoading = false;
      })
      .addCase(logOutUser.fulfilled, state => {
        state.users = { name: '', email: '', avatar: '' };
        state.token = '';
        state.theme = 'dark';
        state.isLogined = false;
      })
      .addCase(editUserTheme.fulfilled, (state, action) =>
{
        state.theme = action.payload.data.theme;
      })
      .addCase(getCurrentUser.fulfilled, (state, action) =>
{
        state.token = action.payload.data.token;
        state.isLogined = true;
        state.isGetCurrentUser = false;
        state.isLoading = false;
      })
      .addCase(getCurrentUser.pending, state => {
        state.isLoading = true;
        state.isGetCurrentUser = true;
      })
      .addCase(getCurrentUser.rejected, state => {
        state.isLoading = false;
        state.token = '';
        state.isGetCurrentUser = false;
      })
      .addCase(editUserProfile.pending, state => {
        state.isLoading = true;
      })
      .addCase(editUserProfile.rejected, state => {
        state.isLoading = false;
      })
      .addCase(editUserProfile.fulfilled, (state, action) =>
{
        state.users.name = action.payload.data.user.name;
        state.users.email = action.payload.data.user.email;
        state.users.avatar =
action.payload.data.user.avatarUrl;
        state.users.id = action.payload.data.user._id;
        state.isLoading = false;
      }),
});

export const authenticationReducer =
authenticationSlice.reducer;

import { createAsyncThunk } from '@reduxjs/toolkit';
```