ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «РОЗРОБКА WEB-ДОДАТКУ ДЛЯ СТВОРЕННЯ СТРАТЕГІЇ ВИКОНАННЯ ТА ВІДСТЕЖЕННЯ ПРОГРЕСУ ПОСТАВЛЕНИХ ЦІЛЕЙ НА ОСНОВІ БІБЛІОТЕКИ REACT.JS»

Виконав: студент 4 курсу, групи ПД-44

спеціальності

121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

Павленко Максим Вадимович.

(прізвище та ініціали)

Керівник Золотухіна О.А.

(прізвище та ініціали)

Рецензент _____

(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти – «Бакалавр»
Напрям підготовки – 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного
забезпечення
_____ Негоденко О.В.
“ ____ ” _____ 2022 року

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

Павленку Максиму Вадимовичу
(прізвище, ім’я, по батькові)

1. Тема роботи: Розробка web-додатку для створення стратегії виконання та відстеження прогресу поставлених цілей на основі бібліотеки React.js
Керівник роботи Золотухіна Оксана Анатоліївна к.т.н., доцент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «16» лютого 2022 року № 22 .
2. Строк подання студентом роботи «3» червня 2022 року
3. Вхідні дані до роботи:
Наукова література з дисцпліни самоменеджменту та дослідження існуючих додатків для самоменеджменту

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
4.1 Дослідження існуючих додатків для самоменеджменту
4.2 Розробка структури додатку для самоменеджменту з метою покращення аналізування та групування поставлених задач
4.3 Написання коду додатку

5. Перелік демонстраційного матеріалу:
5.1 Аналоги
5.2 Таблиця порівнянь аналогів
5.3 Мета, об’єкт та предмет дослідження
5.4 Технічні завдання

5.5 Структура матриці
5.6 Програмні засоби реалізації
5.7 Схема бази даних
5.8 Методи та класи програми
5.9 Екранні форми додатку
5.10 Сторінка стастики додатку
5.11 Апробація результатів дослідження
5.12 Висновки

6. Дата видачі завдання «11» квітня 2022р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	11.04.2022 - 12.04.2022	
2	Аналіз та дослідження існуючих аналогів	13.04.2022 -15.04.2022	
4	Проектування архітектури програмного засобу	16.04.2022 – 20.04.2022	
5	Моделювання об’єкту проектування	21.04.2022 - 22.04.2022	
6	Вступ, висновки, реферат	23.04.2022 – 25.04.2022	
7	Написання 1 розділу	26.04.2022 – 29.04.2022	
8	Написання 2 розділу	30.04.2022 – 04.05.2022	
9	Розробка функціоналу додатку для самоменеджменту	05.05.2022 – 15.05.2022	
10	Написання 3 розділу	16.05.2022 – 22.05.2022	
	Написання 4 розділу	23.05.2022 – 27.05.2022	
11	Розробка демонстраційних матеріалів	28.05.2022	
12	Попередній захист роботи	1.06.2022	
13	Здача роботи	3.06.2022	

Студент _____ Павленко М.В.
(підпис) (прізвище та ініціали)
Керівник роботи _____ Золотухіна О.А.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 60 с, 31 рис, 20 джерел.

Об’єкт дослідження - процес створення та аналізування особистого планування задач.

Предмет дослідження - програмне забезпечення для створення особистого планування задач та контролю якості планування.

Мета роботи - спрощення процесу створення особистого структурованого плану задач за рахунок додавання показників якості планування на основі бібліотеки React.js

Проведено аналіз існуючих додатків для самоменеджменту, з метою виявлення недоліків. Зроблено огляд програмних засобів для побудови плану особистого тайм менеджменту. Серед програмних засобів увага приділялася додаткам які мають чітку структуру побудови планування. Визначено їх основні переваги та недоліки. Визначено головні потреби користувачі при роботі з додатками для створення та аналізу задач з самоменеджменту Проведено огляд засобів програмної реалізації додатку. Описано переваги використання платформи Firebase при розробці serverless додатків та особливості використання платформи.

Розроблено архітектуру додатку. Клієнтська частина реалізована на основі бібліотеки react.js. Серверна частина реалізована на основі сервісу firebase. База даних створена на основі realtime databse від firebase. Розроблений додаток надає можливість створювати та групувати задачі за принципом матриці Ейзенхауера. Ключовими особливостями додатку є перегляд задач у вигляді матриці розподілення задач по критеріям важливості та терміновості

САМОМЕНЕДЖМЕНТ, ПЛАНУВАННЯ, СТАТИСТИКА ЗАДАЧ.

ЗМІСТ

ВСТУП

Актуальністю роботи являється можливість планування задач на основі методу планування за матрицею Ейзенхауера.

Коли людину оточує все більше й більше задач головною проблемою стає правильне розподілення часу на виконання задач. Тримаючи все в голові складно вирішитись що ж робити спочатку, які справи наразі не важливі а що потрібно виконати терміново.

Для вирішення цієї проблеми використовуються додатки для самоменеджменту. Головною ціллю є запис задач до календарю та створення планів на день. Важливим фактором є фільтрування поставлених задач.

Завданням роботи є розробка додатку для створення планування особистого тайм-менеджменту.

Розроблений додаток надає змогу перегляду групованих задач для розуміння повної картини процесу планування. Для зручного аналізу задач було обрано створення плану за допомогою матриці Ейзенхауера, що може допомогти сфокусуватись тільки на потрібних задачах. Також в додатку користувач може планувати задачі у вигляді календарю

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ПЛАНУВАННЯ ТА ВІДСТЕЖЕННЯ ПРОГРЕСУ ПОСТАВЛЕНИХ ЦІЛЕЙ

1.1. Огляд потреб користувачів при роботі з додатками для планування

В сучасному світі додатки для планування задач відіграють все більшу роль в плануванні робочих завдань та особистих справ. Збільшення можливостей викликає зростання кількості можливих справ. Щодня нас оточує купа справ, робота, хобі або ж вивчення нових мов. В списку подій також знаходяться рутинні задачі які відволікають нас від поставленого плану.

Особливу роль в продуктивному виконанні поставлених задач займає фактор відволікання на найменші справи та зазвичай які не приносять користі. Не маючи чіткого плану перед собою ми можемо відволікатись на соціальні мережі, занурюватись в думки, спілкуватись по телефону або ж зовсім перейти до виконання справ які не були заплановані.

Головне завдання планування – розбити одну велику задачу на дрібні, поставити чіткі строки виконання більш дрібних задач, та мінімізувати фактори відволікання.

Попри це, поставши собі багато різних задач, у людини можуть виникати проблеми з їх закриттям у відведений термін. Один із факторів це пріоритетність. Її потрібно вміти розставляти таким чином, щоб в першу чергу були виконані дійсно важливі і термінові завдання. Допомогти з цією проблемою зможе матриця Ейзенхауера. (рис.1.1).

Сенс матриці Ейзенхауера полягає в оцінці конкретного завдання по параметрам важливості та терміновості і сортування списку справ за цим принципом.

Чим більше справ в блоці «А» важливі та термінові завдання, тим більше було не виконано задач з інших блоків своєчасно. В цьому розділі при правильному плануванні не повинно знаходитись будь-яких задач.

Рисунок 1.1 – Матриця Ейзенхауера

Основним блоком є блок «В» Важливі але не термінові завдання . Завдяки тому що завдання в цьому блоку є не терміновими ми можемо спокійно та розмірено виконувати їх. Що призводить до більш якіснішого результату. В цей блок можуть входити як справи заплановані на довгостроковий період так і справи заплановані на день. До цього розділу може відноситись отримання нової професії. Це не термінове завдання, проте воно є головним для досягнення поставленої великої цілі. Задачі будуть виконуватись послідовно та без поспіху що зазвичай є головним фактором успіху поставленої цілі. Проте якщо занадто довго не виконувати завдання з цієї категорії то вони переходять до блоку «А».

Блок не важливих але термінових завдань позначається літерою «С», до цього блоку входять всі справи які не наближають нас до поставлених цілей. Цей список може складатися з перевірки пошти, перегляду купи новин, зустрічей, покупки нових речей. Ці справи повинні виконуватися в вільний час. Більшість задач з цього списку потрібно намагатись делегувати якщо це можливо.

Блок D. Група задач яких ми повинні уникати. Це не важливі та не термінові справи. До нього відноситься все що є відволікаючим фактором: Скролінг соціальних мереж, розмови ні про що по телефону або ж в месенджерах, перегляд розважального контенту на YouTube або ж серіалів. Через те що ці задачі приносять людині швидке отримання дофаміну. Дофамін - гормон який впливає на мотивацію людини та на можливість фокусування на завданнях. Через виконання великої кількість задач цієї категорії у людини зменшується бажання виконувати задачі інших категорій матриці Ейзенхауера. Це пов’язано з тим що при виконанні задач з більш важливих категорій потрібно більше часу для отримання організмом дофаміну. Саме через це потрібно намагатись уникати потрапляти в категорію D, або хоча б мінімізувати заняття цими справами.

Розклавши задачі по пріоритетності за допомоги матриці Ейзенхауера людина може встигнути зробити більше через відкидання не важливих справ та сфокусуватись на тому що являється дійсно важливим.

1.2. Огляд існуючих додатків для планування задач

Засоби для самоорганізації можуть бути описані наступним переліком:

- планування в думках;
- створення списку задач в блокноті;
- швидкі замітки в телефоні;
- додатки для планування.

Планувати всі свої справи в думках найлегший проте най не ефективніший спосіб. Людина не може тримати всі справи в голові та провести чіткий аналіз, відкинувши те що не є ефективним. Також є шанс що деякі справи можуть загубитися в думках.

Записуючи свої плани в блокноті є можливість переглянути їх, провести оцінку та відредагувати. Проте для перегляду своїх планів потрібно увесь час мати під рукою блокнот з планами що є не надто зручним. Також щоб змінити плани

доведеться переписувати весь список або ж креслити послідовність що може призвести до нечитабельності планів.

Планування в швидких замітках на мобільному пристрої допомагає не тримати в голові купу справ та надає можливість зручно редагувати. Якщо задачі стосуються чого швидкого та не потребують глибокого аналізу це оптимальний варіант. Проте якщо потрібно якісно запланувати послідовність справ то можуть виникнути проблеми через обмежену ширину пристрою що може викликати певні незручності. Також замітки в телефоні не надають можливості проведення аналізу виконаних справ або розставлення приорітетів.

Trello надає можливість не пропускати навіть найдрібніші деталі завдань проекту або ж особистих задач. Головними особливостями даного додатку є наявність канбан дошок. Дошка Kanban — це інструмент для візуалізації робочих процесів, його головню ідеєю є допомога у внесені ясності в робочий процес та підвищення ефективність, обмежуючи незавершену роботу.

Kanban (англ.: вивіска) був створений як візуальна система планування, він був основою планування виробничої системи Toyota. У 2007 році Девід Андерсон розвинув ідею методу Канбан і представив дошку Канбан. Колега Андерсона, Даррен Девіс запропонував візуалізувати робочий процес за допомогою дошки.

Таким чином була створена дошка Kanban, яка відома на сьогоднішній день, ставши одним із найзручніших інструментів управління проектами для робочих процесів. Нині її використання методологією Agile настільки широке, що часто можна почути, як люди називають дошки Kanban дошками для гнучких завдань.

Приклад використання Kanban дошки в робочому проекті (рис. 1.2). Завдяки інтуїтивно зрозумілому дизайну Trello користувачі можуть без проблемно користуватися додатком без досвіду використання програмами для керування задач. В Trello досить легко створити індивідуальні робочі процеси під різні проекти та розділити проект по групам задач за своїм вподобанням.

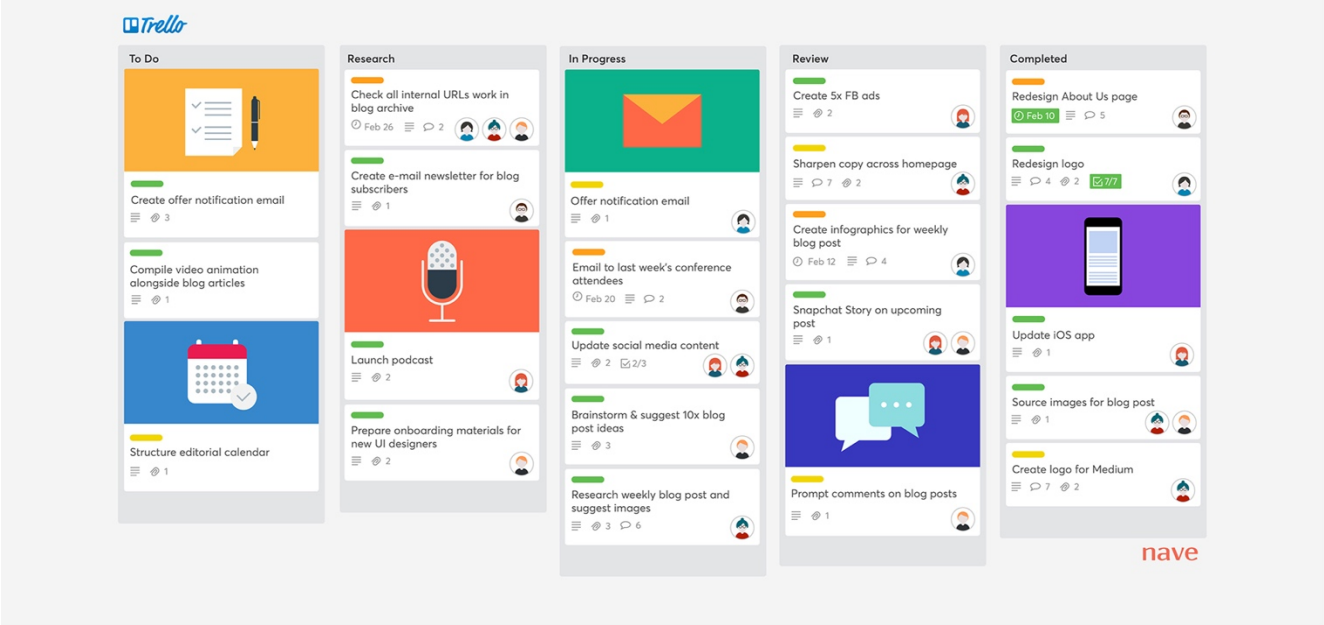


Рисунок 1.2 – Kanban дошка

Для управління проектом в сфері розробки може бути створена дошка (рис. 1.3) з наступними колонкам:

- Задача;
- В роботі;
- В тестуванні;
- Виконано.

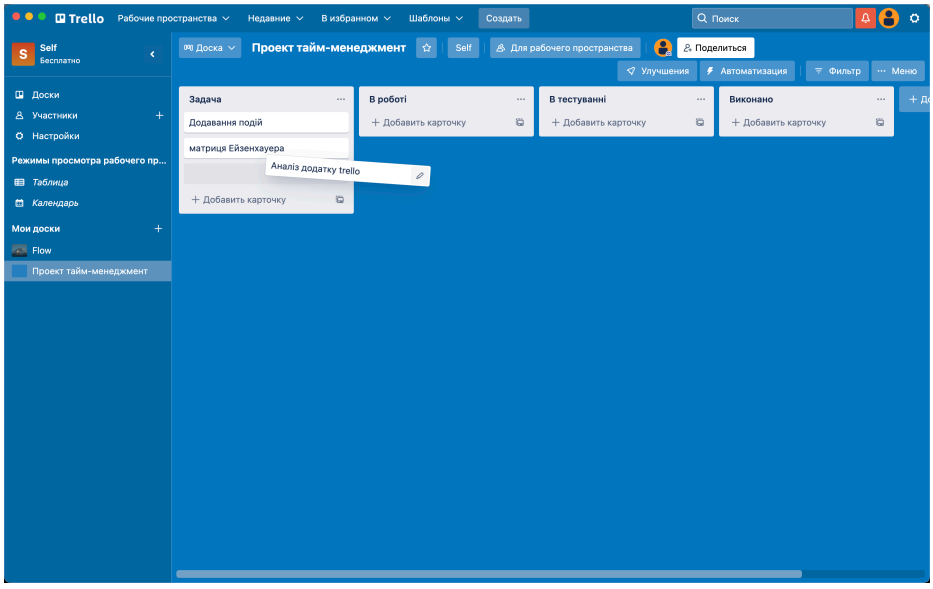


Рисунок 1.3 – Приклад зміни статусу задачі

Щоб змінити статус задачі її потрібно перетягнути в відповідну колонку. Учасники проекту одразу можуть помітити зміни та на основі них брати в роботу інші завдання, що дуже спрощую комунікацію між розробниками проекту .

Головною відмінністю Trello від аналогів є велика різноманітність кастомізування карток. Функціонал картки надає змогу:

- додавати інших учасників на карту для сповіщення їх про будь які зміни а також для розуміння хто саме відповідальний за завдання;
- додавати теги на картку;
- додавати чек-бокси з будь яким надписом;
- додавати чіткі строки на виконання завдання;
- додавати вкладення, збережені на власному пристрої або ж взяті з гугл диска;
- додавати графіки.

Trello дуже простий для інтеграції зі сторонніми додатками, у користувачів є можливість додати віджет гугл диску до картки, або ж додати гілку з GitHub. Провести голосування в картці, під`єднати Hangoust Chat до Trello, все це включено в додаток та є безкоштовним. Trello – ідеальний варіант для менеджменту проекту але не надто зручний для само менеджменту.

Any.DO - це веб-додаток зі списком справ і інструментами керування завданнями з синхронізацією даних між різними пристроями. Додаток доступний як безкоштовна персональна версія, а також як платна з розширеним наобором функцій. За допомогою Any.do користувачі можуть створювати списки справ і переглядати свої майбутні та виконані завдання в одному місці (рис. 1.4).

На головній сторінці користувачу відображається список його задач розділений на 4 категорії. Натиснувши на кнопку «+» нам відображається екран з створенням нової задачі. В якій вже є перелік найбільш використовуваних категорій, таких як:

- зателефонувати;
- перевірити;

- отримати;
- зробити;
- забрати.

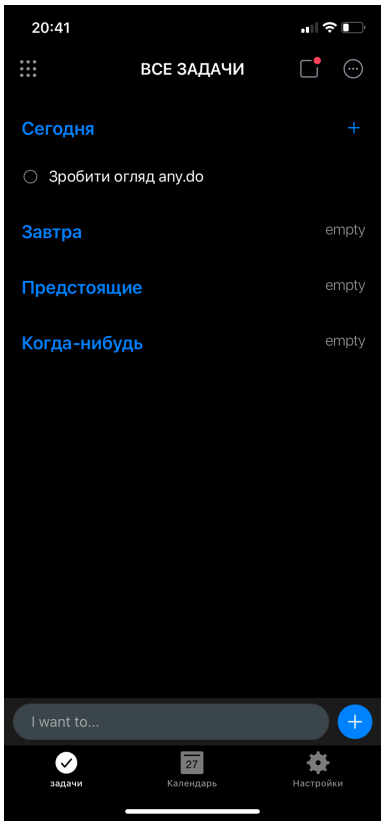


Рисунок 1.4 – Головний екран додатку

В Any.do можна створювати як одноразові, так і повторювані завдання, а користувачі можуть встановлювати нагадування про завдання, які запускаються за датою, часом або місцем розташування. Завдяки повторюваному завданню користувачі можуть вибирати із звичайних частот, включаючи щоденні, щотижневі та щомісячні, або налаштовувати власні індивідуальні налаштування для повторення. Any.do також містить ярлики дій, пов'язані із завданнями, що дає користувачам можливість здійснювати дзвінки, надсилати електронні листи чи текстові повідомлення, а також робити замовлення чи покупки з програми. Any.do пропонує інтеграцію зі сторонніми календарними інструментами, такими як Календар Google і Outlook (рис. 1.5).

Рисунок 1.5 – Синхронізований календар google

Він також доступний через мобільний додаток для Android та iOS, що дає користувачам доступ до свого щоденника, списку покупок або списку справ з будь-якого місця.

Одним із плюсів є змога додавання віджету на головний екран на пристроях IOS що може бути корисним для швидкого переміщення задач з голови в чіткий список на пристрої.

Також додаток має функцію фокусування на задачі за методикою pomodoro, яка допомагає користувачу справлятися з фактором відволікання. Проте ця функція доступна лише за наявності платної підписки.

До мінусів додатку відноситься те що задачі в основному відсортовані лише за датами їх виконання, створюючи велику кількість задач користувач втрачає фокус з найголовніших. Також для повноцінного використання потрібно сплачувати підписку яка коштує 5.99$ на місяць що може знижувати його популярність.

Evernote – Найпопулярніший блокнот який з часом перетворився на зручний таск-менеджер. З моменту запуску бета-версії в 2008 році понад 200 мільйонів користувачів використовували Evernote. Завдяки великій кількості вже готових до використання шаблонів користувач легко може обрати потрібну структуру примітки під свої потреби (рис. 1.6).

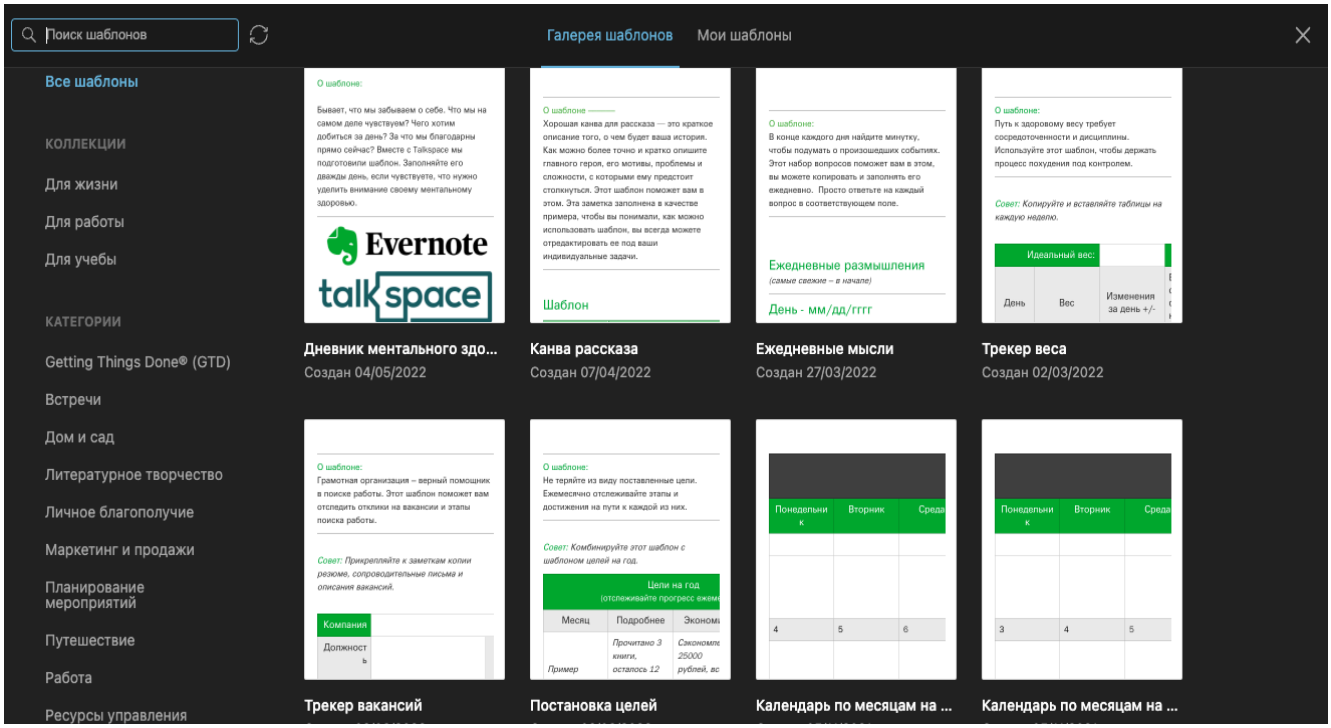


Рисунок 1.6 – Фрагмент списку шаблонів Evernote

Evernote має розширені параметри, такі як оптичне розпізнавання символів (OCR), сканування документів і функції керування командою, щоб відстежувати кожну нотатку, кожен завантажений файл і всі терміни членів вашої команди. Однак це стосується лише платних версій. Безкоштовно використовуючи Evernote ви не матимете цих функцій.

Evernote має безліч функцій, які чітко відрізняють його від інших додатків для створення нотаток. При правильному використанні він може бути надійним інструментом управління проектами, до якого вся команда може отримати доступ практично з усіх своїх пристроїв, які потім можна синхронізувати з хмарою. Це

виключає втрату даних, оскільки кожен член команди може підключитися та використовувати додаток у будь-який час.

Існує кілька типів нотаток і файлів, які можна створювати, зокрема аудіозаписи, фотографії, рукописні каракулі, вкладення та нагадування. Звичайно, оскільки Evernote — це програма для створення нотаток, сама нотатка є основною одиницею, для якої присвячено більшість опцій та налаштувань.

Ще однією особливістю додатку є те що всі замітки користувача знаходяться в одному списку з задачами поруч з яким завжди відкритий редактор (рис 1.7).

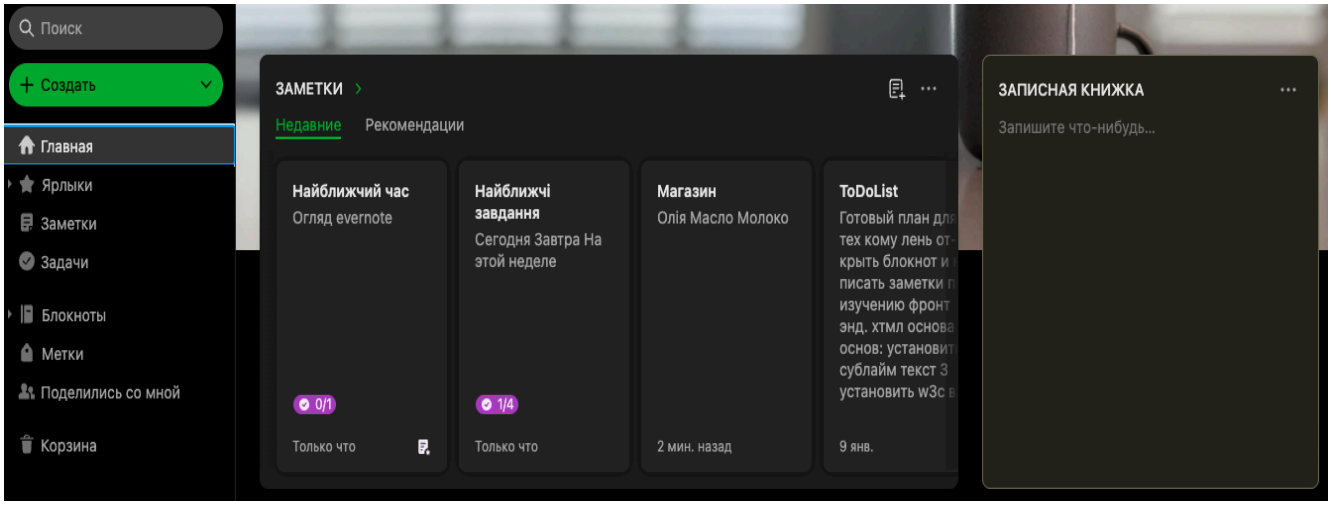


Рисунок 1.7 – Головна сторінка Evernote

На головній сторінці знаходиться список всіх карток з примітками користувача та компонент для створення швидких приміток. Якщо картка відноситься до типу «Задачі» то в нижньому лівому куту знаходиться показник з загальною та виконаною кількістю поставлених задач. Користувачу доступна можливість створення карток з малюнком який він може зробити мишкою або стилусом на планшеті. Що є дуже корисним при створенні особливих приміток коли простіше створити блок схему або ж намалювати графік ніж описувати ідею текстом.

Додаток має шаблони методології планування «Getting Things Done» що є корисним в аналізі підходу до життя та роботи.

Таблиця 1.1 - Зведені результати аналізу характеристик додатків для тайм-менеджменту

Показник платформи	Trello	Evernote	Any.DO
Створення задач на день	-	+	+
Розстановки пріоритетів	-	-	+
Групування задач за власними категоріями	+	+	-
Нагадування про задачі	-	+	+
Виділення кольором	+	+	-
Кросплатформенність	+	+	+
Інтеграція зі стороннім календарем	+	-	+
Інше	Широка можливість додавання віджетів до проекту	Можливість додавання малюнків в примітки. Наявність шаблонів планування	Наявність віджету на пристроях ios, Режим фокусування

1.3. Вибір засобів програмної реалізації додатку

Додаток для створення особистого тайм менеджменту розробляється в першу чергу як десктопна версія у вигляді односторінкового додатку. Для цих цілей

використовується бібліотека React.js. Для покращення завантажуваності додатку використовуються service workers. Сервером виступає firebase що дуже зручно як для побудови бази данних так і для хостингу та тестування фронтенд частини.

1.3.1 React

React — це бібліотека JavaScript, створена Facebook. Це найпопулярніша бібліотека для створення односторінкових програм та інтерактивних інтерфейсів користувача. React був створений в 2011 році Джорданом Волке в Facebook. На той час платформа виросла настільки, що існуючих рішень виявилося недостатньо, тому вони створили власну фронтенд-фреймворк (або бібліотеку), щоб задовольнити свої зростаючі потреби. У 2013 році React був з відкритим кодом, і з тих пір він зріс у геометричній прогресії, постійно додаючи нові функції, а навколо нього розвивається дивовижна спільнота розробників.

React є досить зрілим і перевіреним часом, і його використовують багато відомих компаній (Netflix, AirBnB, Instagram, і це деякі з них. Також Facebook). Він легший і продуктивніший, ніж Angular, який є рішенням Google для створення зовнішніх програм. React також не нав’язує будь-які конкретні способи розробки і дозволяє вибрати з величезного набору додаткових бібліотек. Єдиного «способу реакції» не існує. Таким чином, можна створити власний стек React відповідно до потреб проекту.

Основною структурною одиницею в React є компонент - будівельний блок додатка. Це може бути що завгодно — спливаюче вікно, кнопка, піктограма, уривок тексту або контейнер, щоб утримувати все це всередині та передавати деякі дані кожному. React потребує надати інформацію про те, як мають виглядати всі компоненти залежно від конкретних параметрів (також відомих як «стан» програми), і вносить відповідні зміни у подання відповідно до змін у стані. Це прискорює та полегшує розробку навіть найскладніших інтерфейсних програм.

React також дуже ефективний у внесенні змін до уявлень, оскільки він віртуалізує вміст усієї програми і спочатку вносить зміни до цього віртуального «подання», а потім визначає найкращий і найшвидший спосіб відобразити зміни на

екрані. Це призводить до дуже гладких та продуктивних програм, які забезпечують чудовий досвід користувача.

1.3.2 Firebase

Firebase — це платформа для розробки, спочатку відома своєю базою даних у реальному часі, яка все ще є основою багато вузлової бази даних ключ-значення, оптимізованої для синхронізації даних, часто між машинами користувачів або смартфонами та централізованим сховищем у хмарі. Він розроблений для полегшення процесу розробки, обробляючи більшу частину передавання та витягування даних. Це спрощує процеси, пов’язаніз керуванням версіями або розташуванням. Таким чином, це бекенд-як-сервіс (Baas). Він надає розробникам різноманітні інструменти та послуги, які допомагають їм розробляти якісні програми, розширювати базу користувачів та отримувати прибуток. Він побудований на інфраструктурі Google.

Ключові особливості firebase:

- Аутентифікація - він підтримує аутентифікацію за допомогою паролів, номерів телефонів, Google, Facebook, Twitter тощо. Аутентифікацію Firebase (SDK) можна використовувати, щоб вручну інтегрувати один або кілька методів входу в програму.

- База даних реального часу - Дані синхронізуються між усіма клієнтами в режимі реального часу і залишаються доступними, навіть коли програма переходить в автономний режим.

- Хостинг - Firebase Hosting забезпечує швидкий хостинг для веб-програми; вміст кешується в мережах доставки вмісту по всьому світу.

- Випробувальна лабораторія - Додаток тестується на віртуальних і фізичних пристроях, розташованих у центрах обробки даних Google.

- Повідомлення - Повідомлення можна надсилати за допомогою firebase без додаткового кодування.

1.3.3 Безсерверна модель розробки

Безсерверна модель розробки на основі хмари, яка дозволяє розробникам створювати та запускати програми без необхідності керувати серверами. Хмарний постачальник виконує рутинну роботу з надання, підтримки та масштабування інфраструктури сервера. Розробники можуть просто запакувати свій код у контейнери для розгортання. Після розгортання безсерверні програми реагують на попит і автоматично збільшуються та зменшуються за потреби. Безсерверні пропозиції від постачальників загальнодоступних хмар, як правило, надаються на вимогу за допомогою моделі виконання на основі подій. В результаті, коли безсерверна функція простоює, вона нічого не коштує. Безсерверна відмінність від інших моделей хмарних обчислень полягає в тому, що постачальник хмар відповідає за управління хмарною інфраструктурою і масштабуванням програм. Безсерверні програми розгортаються в контейнерах, які автоматично запускаються на вимогу при виклику.

Відповідно до стандартної моделі хмарних обчислень типу інфраструктура як послуга (IaaS) користувачі попередньо купують одиниці потужності, тобто ви платите постачальнику загальнодоступної хмари за постійні серверні компоненти для запуску ваших програм. Відповідальність користувача за збільшення потужності сервера під час високого попиту та зменшення, коли ця потужність більше не потрібна. Хмарна інфраструктура, необхідна для запуску програми, активна, навіть коли програма не використовується.

На відміну від безсерверної архітектури, програми запускаються лише за потреби. Коли подія запускає код програми, постачальник загальнодоступної хмари динамічно виділяє ресурси для цього коду. Користувач припиняє платити, коли код закінчується. Безсерверний режим звільняє розробників від рутинних і дрібних завдань, пов’язаних із масштабуванням програми та наданням серверів.

Безсерверні рутинні завдання, такі як керування операційною системою та файловою системою, виправлення системи безпеки , балансування навантаження, керування ємністю, масштабування, ведення журналів і моніторинг, передаються постачальнику хмарних послуг.

Можна створити повністю безсерверну програму або програму, що складається з частково безсерверних і частково традиційних компонентів мікросервісів .

1.3.4 Service workers

Service workers - є основною частиною PWA. Вони забезпечують швидке завантаження (незалежно від мережі), офлайн-доступ, push-повідомлення та інші можливості.

Користувачі очікують, що програми запускатимуться при повільних або нестабільних мережевих з'єднаннях або навіть у автономному режимі. Вони очікують, що вміст, з яким вони нещодавно взаємодіяли, наприклад медіа-треки або квитки та маршрути, буде доступним і придатним для використання. Коли запит неможливий, вони очікують, що програма повідомить їм замість того, щоб мовчазно вийти з ладу чи збою. І користувачі хочуть зробити все швидко. Підводячи підсумок: користувачі очікують, що PWA будуть надійними.

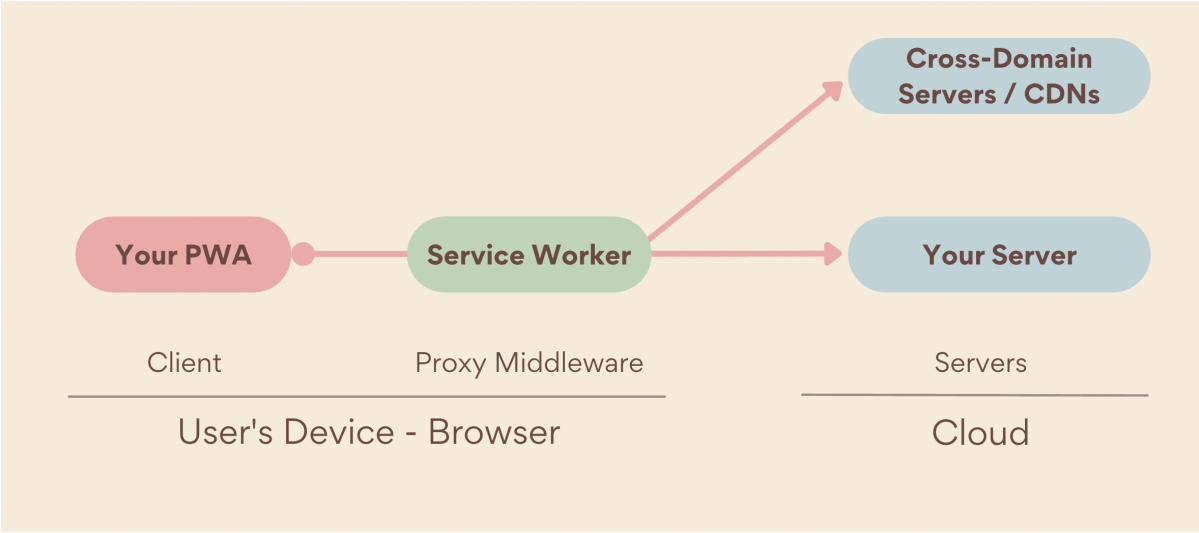


Рисунок 1.8 – Процес роботи Service worker

Коли програма запитує ресурс, який охоплює сферу дії сервіс-воркера, у тому числі коли користувач перебуває в автономному режимі, сервіс-воркер перехоплює запит, діючи як мережевий проксі-сервер. Потім він може вирішити, чи повинен

він обслуговувати ресурс із кешу через Cache Storage API, із мережі, як це зазвичай відбувається без сервісного працівника, чи створити його за допомогою локального алгоритму. Це дає змогу надавати досвід, подібний до того, що надається додатком платформи. Він може працювати навіть повністю в автономному режимі. Процес роботи наведений на схемі (рис 1.8.)

Сервісні працівники мають життєвий цикл, який визначає, як вони встановлюються, це окремо від вашої інсталяції PWA. Життєвий цикл Service Worker починається з реєстрації Service Worker. Потім браузер намагається завантажити та проаналізувати файл Service Worker. Якщо синтаксичний аналіз вдається, його подія встановлення запускається. Подія встановлення запускається лише один раз.

Встановлення Service Worker відбувається непомітно, без дозволу користувача, навіть якщо користувач не встановлює PWA. API Service Worker доступний навіть на платформах, які не підтримують встановлення PWA, таких як Safari та Firefox на настільних пристроях.

Після встановлення сервісний працівник ще не контролює своїх клієнтів, включаючи ваш PWA. Спочатку його потрібно активувати. Коли сервісний працівник буде готовий контролювати своїх клієнтів, «activate» подія запускається. Однак це не означає, що сторінка, на якій був зареєстрований сервісник, буде керованою. За замовчуванням він не перейме контроль до наступного разу, коли ви перейдете на цю сторінку, через перезавантаження сторінки або повторне відкриття PWA.

1.3.5 Progressive web application

Термін «Progressive web application» придумав інженер Google Chrome Алекс Рассел, який використав його для опису нового покоління веб-додатків, які завантажуються так само, як і звичайні веб-сайти, але користуються перевагами функцій, які підтримуються сучасними браузерами, зокрема сервісних працівників та маніфестів веб-програм, щоб запропонувати користувачам такі функції, як

робота в автономному режимі, push-повідомлення та інші функції, які традиційно пов’язувалися лише з нативними програмами.

Таким чином, PWA ефективно долають розрив між мобільними додатками та веб-сайтами, пропонуючи найкраще з обох сторін. PWA корисні в той час, коли користувачі вимагають зручного мобільного користування, але перевантажені кількістю програм на своїх пристроях, що змушує їх неохоче встановлювати нові.

Багато провідних компаній вже перехопили переваги PWA і випустили власні веб-додатки з вбудованою функціональністю. Усі основні переваги та недоліки прогресивних веб-додатків випливають з того факту, що PWA об’єднують зручність і охоплення сайтів з функціональністю рідних мобільних додатків. PWA можна кешувати веб-браузером і використовувати навіть у автономному режимі.

Оскільки PWA використовують так звані сервісні працівники, які є файлами JavaScript, які запускаються окремо від основного потоку веб-браузера та проактивно контролюють кешування, вони можуть забезпечити набагато кращу продуктивність, ніж традиційні веб-сайти.

1.4. Постановка задач дипломної роботи

В ході аналізу програмного забезпечення для особистого тайм менеджменту виявлено що популярні додатки не забезпечують користувача можливістю перегляду аналітики якості планування задач та не мають чіткого методу за яким можливо легко відфільтрувати задачі за показниками важливості та терміновості.

Створення додатку який буде надавати користувач можливість швидкого створення задач та миттєвого розподілення їх до одної з чотирьох чітко визначених категорій спростить процес планування особистих цілей на майбутнє та задач які потрібно виконати терміново або ж покаже які задачі можливо делегувати.

Аналіз особливостей формування особистого тайм менеджменту та існуючих засобів для створення ведення структури та порядку особистих справ надає змогу сформувати вимоги майбутнього програмного продукту.

Програма ведення та відстеження статистики особистого тайм менеджменту повинна бути реалізована у вигляді MVP (Minimal Viable Product) як веб додаток.

Провівши аналіз додатків аналогів та дослідивши процес ефективного планування задач було створено основні задачі дипломної роботи:

1. Аналіз потреб користувачів при роботі з додатками для самоменеджменту;
2. Аналіз існуючих додатків з метою виявлення недоліків;
3. Розробка архітектури клієнтської частини на основі бібліотеки React.js;
4. Розробка серверної частини на основі сервісу Firebase;
5. Розробка архітектури бази даних;
6. Реалізація головних модулів додатку:
   - внесення в календар запланованої події;
   - додавання задачі до загального списку;
   - створення фільтрів на основі матриці Ейзенхауера;
   - перегляд кількості важливих задач;
   - сповіщення про надмірну кількість задач із розділу «А»;
   - автоматичне розподілення задачі по групам на основі обраних параметрів;

Мета роботи – Розробка веб-додатку для створення стратегії виконання та відстеження прогресу поставлених цілей на основі бібліотеки React.JS.

Об’єкт дослідження – процес створення та аналізування особистого планування задач.

Предмет дослідження – програмне забезпечення для створення особистого планування задач та контролю якості планування.

Завдання роботи – розробка додатку для створення планування особистого тайм-менеджменту.

Висновки за розділом 1.

1. Проведено аналіз основних задач при створенні персонального тайм менеджменту. Розглянуто методи вирішення проблеми планування. Створено структуру планування задач на основі методу матриці Ейзенхауера.

2. Зроблено огляд додатків з можливістю організування особистого планування задач. Основну увагу було приділено додаткам які мають основні набори для створення таблиць, графіків та можливість відображення даних у вигляді календарю. Проаналізовано переваги та недоліки оглянутих додатків. В результаті аналізу додатків для особистого тайм менеджменту було сформовано вимоги до майбутнього веб додатку.

3. Проведено огляд програмної реалізації додатку. Описано переваги бібліотеки React.js, serverless технології та сервісу Firebase.

4. Сформовано вимоги до веб додатку.

2. РОЗРОБКА СТРУКТУРИ ДОДАТКУ ДЛЯ ТАЙМ-МЕНЕДЖМЕНТУ НА ОСНОВІ БІБЛІОТЕКИ REACT.JS

2.1. Моделювання об’єкту проектування

2.1.1. Діаграма прецедентів системи

Діаграма прецедентів системи (рис. 2.1) являє собою відображення взаємодії акторів з системою. Додаток має одного актора який взаємодіє з системою. Для роботи з додатком користувачі спочатку потрібно авторизуватись. Після вдалої авторизації користувачу стають доступні методи перегляду завдань, та створення нових.

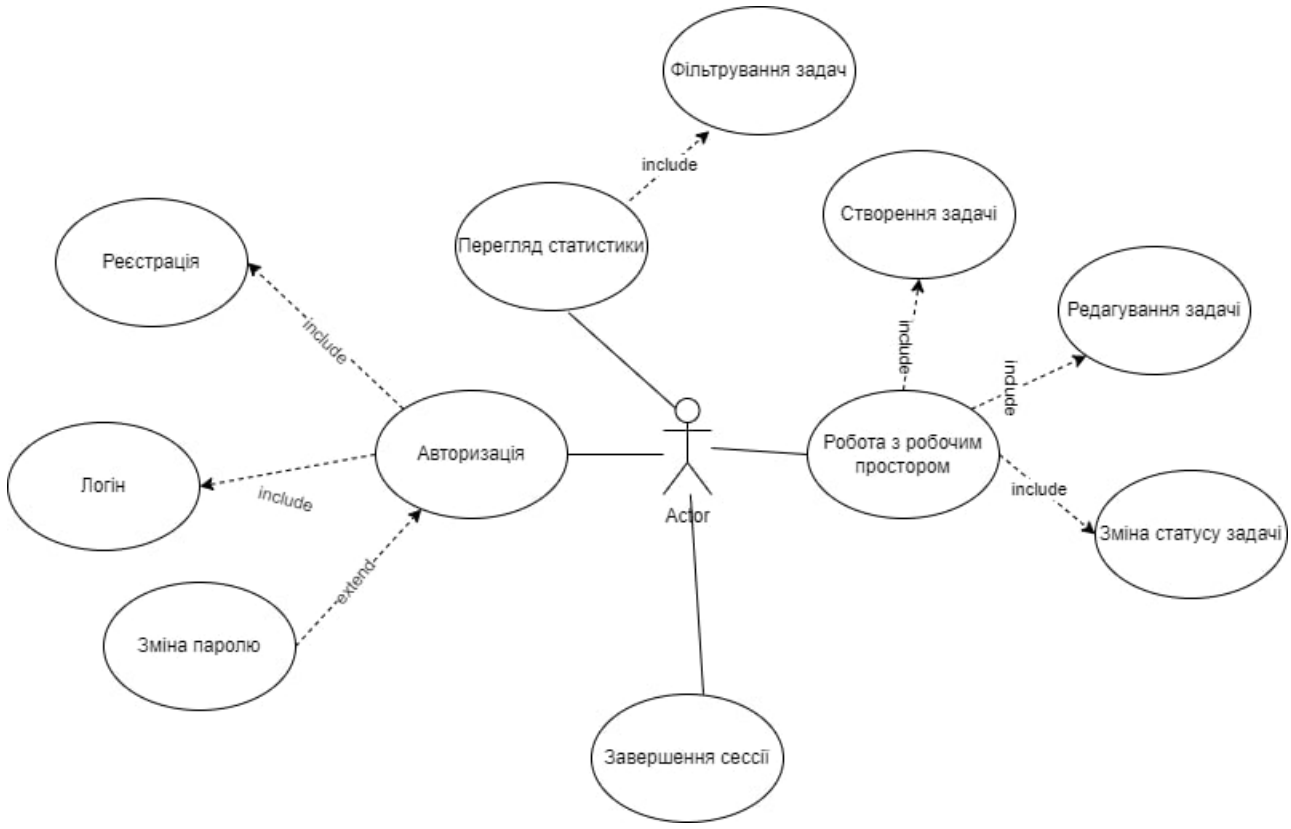


Рисунок 2.1 - UML Діаграма прецедентів системи

2.1.2. Структура даних в системі

Роль бази даних виконувала realtime database від сервісу firebase. Realtime database являє собою нереляційну базу даних у вигляді json об’єкту. Firebase надає можливість встановлювати правила на запис, видалення, оновлення та додавання за умовами авторизації користувача. Структура бази даних з користувачами зображена на рисунку 2.2.

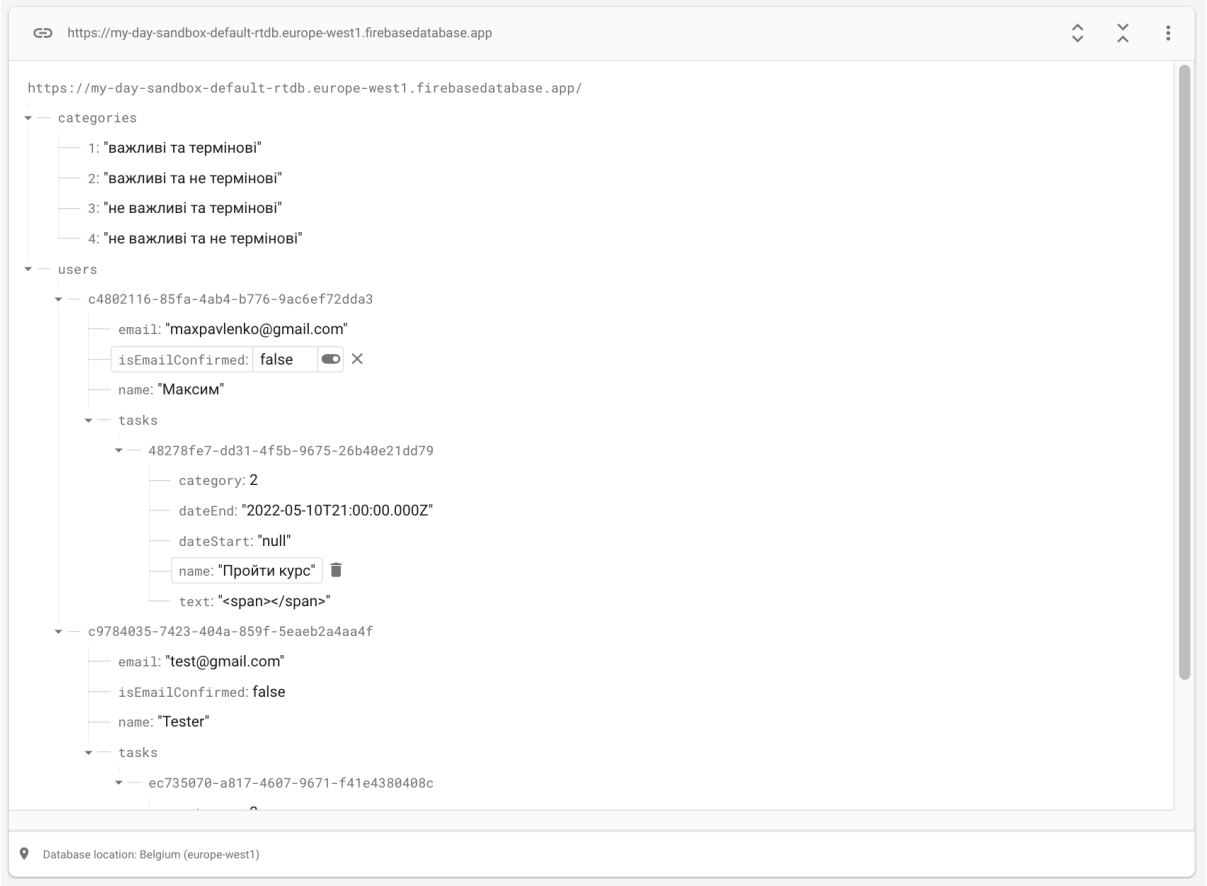


Рисунок 2.2 – Інтерфейс бази даних

Основними сутностями структури є нижченаведені об’єкти.

Categories – Об’єкт з константами категорій задач (рис. 2.3), які користувач має змогу обрати при створенні нової задачі.

Поле id – є ключом таблиці Categories, має числовий тип. Значення формуються автоматично.

Поле name – має тип String та містить інформацію про назву категорії яка відображається користувачу

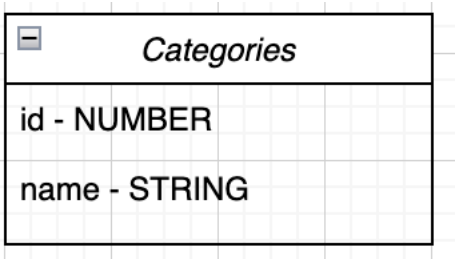


Рисунок 2.3 – Структура об’єкту Categories

User – Головний об’єкт (рис. 2.4), зберігає в собі дані про користувача та об’єкт з масивом задач які відносяться до даного користувача.

Поле id – має тип GUID та містить унікальний ідентифікатор користувача.

Поле name – має тип string, містить ім’я користувача.

Поле email – має тип string, валідується за типом email, містить в собі пошту користувача.

Поле isEmailConfirmed – має тип Boolean, Містить в собі інформацю про підтвердження пошти користувачем

Поле tasks – Зберігає в собі масив об’єктів Task (Рис 2.5)

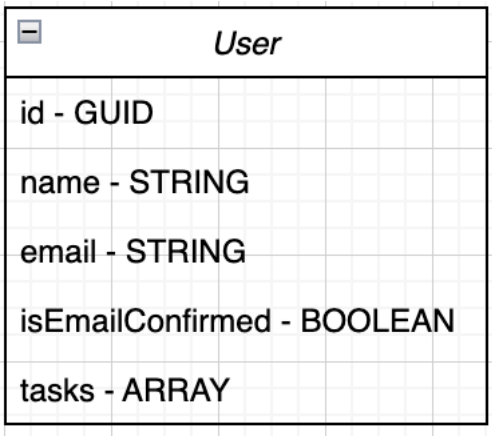


Рисунок 2.4 – Структура об’єкту User

Task – Об’єкт із поставленою задачею користувача (рис. 2.5). Знаходиться в масиві об’єктів Tasks які належать до об’єкту User.

Поле id - має тип GUID та містить унікальний ідентифікатор об’єкту Task.

Поле title – має тип String, містить заголовок завдання, має обмеження в 40 символів

Поле text – має тип String, містить в собі текст в форматі html з детальним описом завдання.

Поле category – має тип string, містить в собі id таблиці Category.

Поле dateStart – має тип DateTime містить в собі дату створення завдання в форматі yyyy-mm-ddThh:mm:ss.000Z.

Поле dateEnd - має тип DateTime містить в собі дату виконання завдання в форматі yyyy-mm-ddThh:mm:ss.000Z.

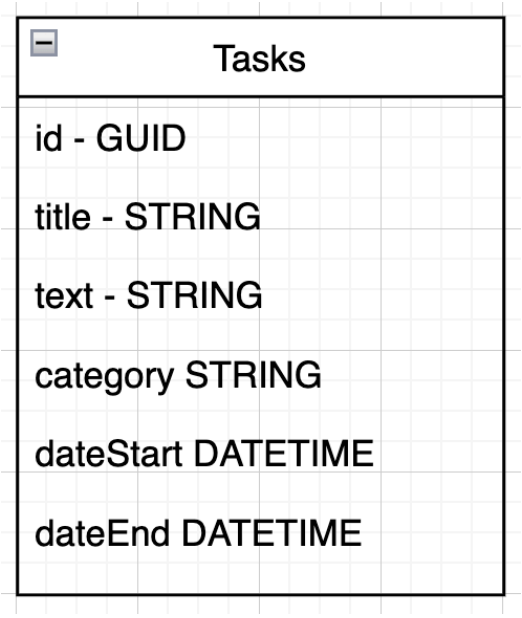


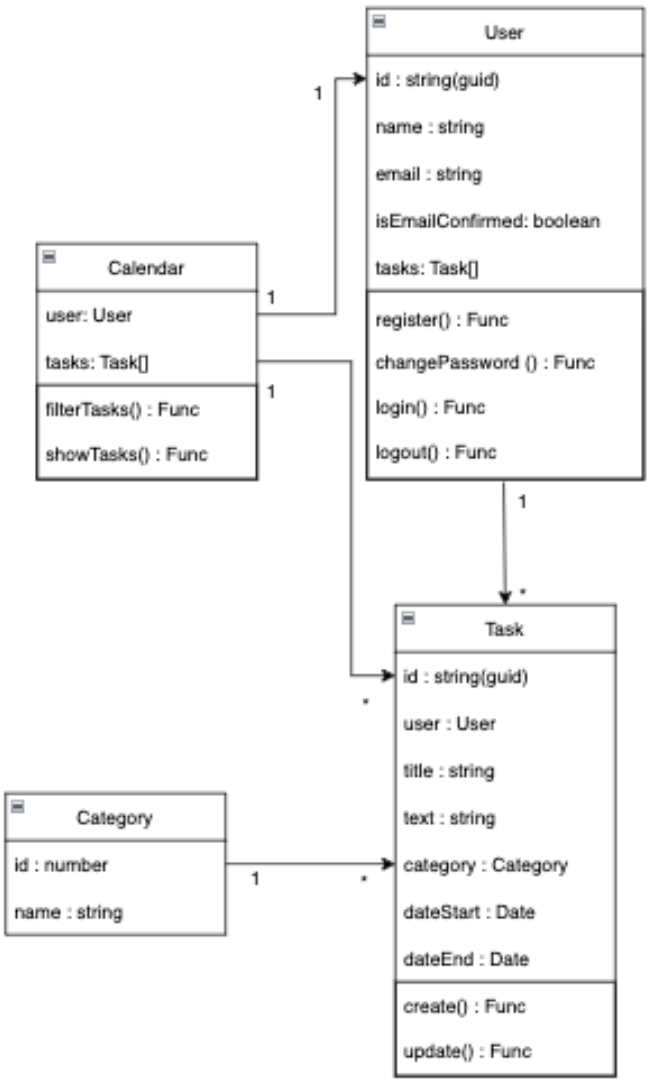
Рисунок 2.5 – Структура об’єкту Task

2.2 Структура системи

2.2.1. Структура клієнтської частини

Клієнтська частина додатку реалізована на основі бібліотеки React.js що надає змогу розробки з компонентним підходом. React має велику кількість бібліотек з готовими компонентами, для побудови UI частини було обрано одну з таких бібліотек а саме rsuitejs.

Структура клієнтської частини має 5 головних класів як зображені на рисунку 2.6.



Клас Calendar містить в собі дані про користувача та про його завдання, містить в собі методи для роботи з календарем.

Клас User зберігає в собі всі данні про користувача. Головним полем є масив об’єктів Task які зберігаються в об’єкті User в структурі бази даних. При створенні нової задачі викликається метод create класу Task, після завершення методу відбувається оновлення календарю за допомогою класу Calendar. Для оновлення стану додатку та відображення нової задачі в інтерфейсі користувача відбувається рендер додатку. Всі дії пов’язані між собою на методології реактивного програмування. Для цього був використаний патерн EventObserver.

Керування станом додатку відбувається за допомогою бібліотеки Redux. Redux оснований на flux архітектурі що надає можливість будь-якому компоненту отримати доступ до даних які зберігаються в одному об’єкті. Архітектура redux зображена на рисунку 2.7.

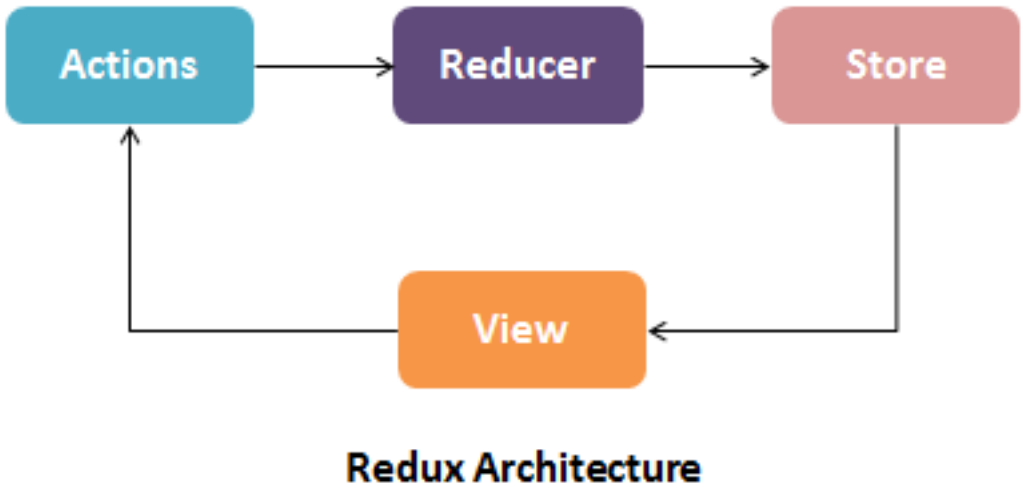


Рисунок 2.7 – Redux архітектура

Потік даних Redux завжди односпрямований, і дуже простий. За дані відповідає сховище, компоненти викликають лише методи оновлення даних і потім отримують оновлені дані.

2.2.2. Структура серверної частини

Серверну роль додатку відіграє firebase. Firebase надає доступ до модулю realtime database а також до модулю аутентифікації. Доступ до редагування даних реалізований у вигляді rest сервісу. За кожний об’єкт таблиці відповідає свій роут.

Модуль аутентифікації надає можливість створювати користувача за допомогою пошти та паролю або ж пропонується авторизуватись через сервіси google. Після авторизацію користувачу надаються права на редагування які обмежені по id користувача.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Оцінка та планування розробки проекту

При створені продукту, на початковому етапі стоїть завдання з визначення зацікавленості користувачів. Уникнення ризиків може забезпечити створення MVP продукту. Основною перевагою MVP являється те що розробник можете зрозуміти інтерес клієнтів до продукту, не розробляючи продукт повністю.

Основною задачею даного підходу є виявлення та створення головних модулів з якими буде взаємодіяти користувач. Підхід структури розробки проекту оснований на методології scrum (рис. 3.1).

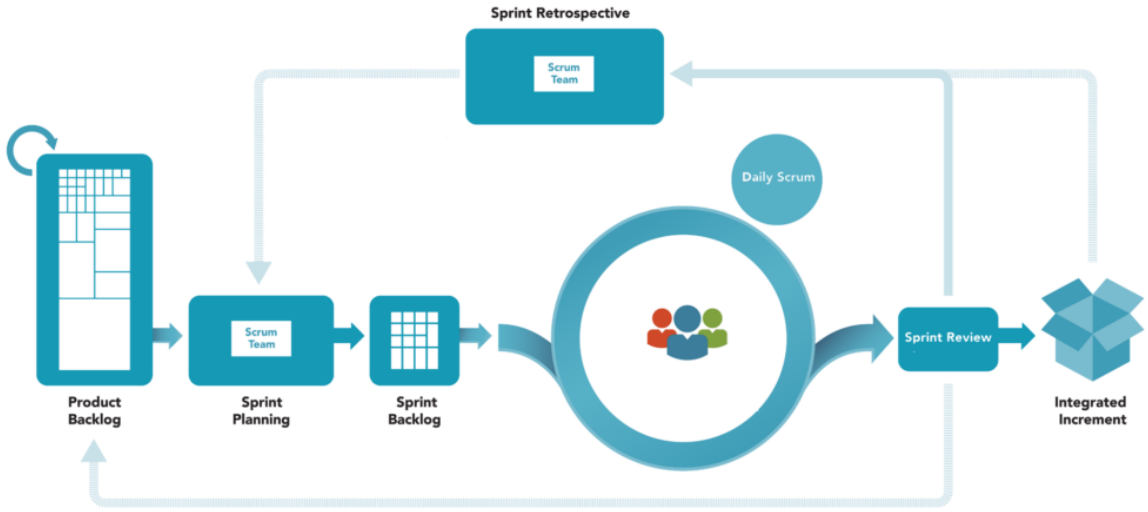


Рисунок 3.1 – Scrum методологія

Методика Scrum за своєю сутністю є евристичною. В її основі лежить постійне навчання та адаптація до мінливих факторів. Згідно з Scrum, команда не знає всього на початку проекту, але розвиватиметься, вивчаючи уроки з досвіду. У структурі Scrum закладена та свобода, з якою команди пристосовуються до змінних умов та вимог користувачів. Робочий процес передбачає зміну пріоритетів та короткі цикли релізу, що сприяє постійному навчанню та вдосконаленню продукту.

Робочий процес був розподілений на спрінти. Приклад першого спрінту зображений на рисунку 3.2

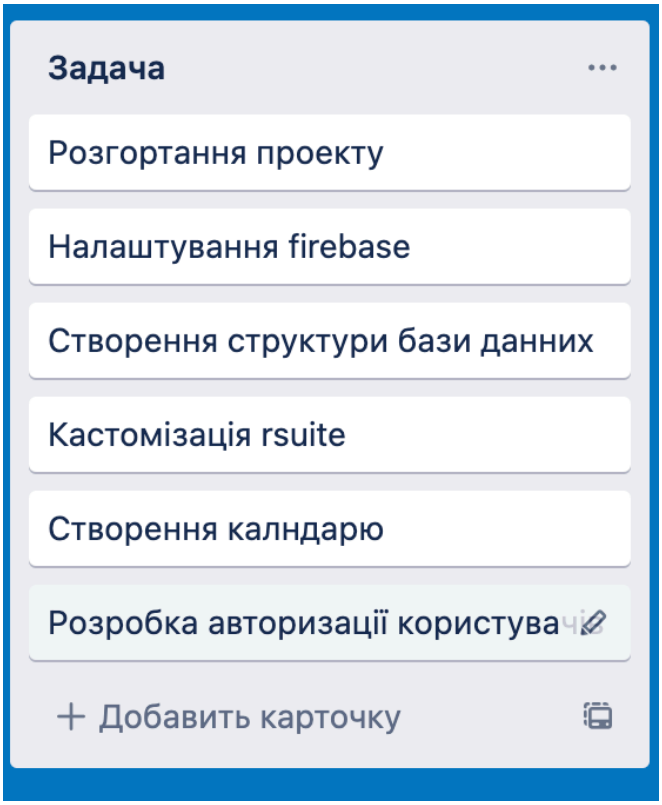


Рисунок 3.2 – Вигляд карток з задачами спринту

Для відстеження виконання задач було обрано інструмент trello. Кожна картка описує окрему задачу що надає змогу аналізувати кількість виконаних задач поставленого спринту. Дошка з картками (Рис. 3.3) задач має наступні колонки:

- Задача – Група задач які потрібно виконати;
- В роботі – Група задач які взяті в роботу та виконуються в даний момент;
- В тестуванні – Задачі які виконані та підлягають тестуванню;
- Виконано – Всі виконані задачі які пройшли етапи розробки та тестування.

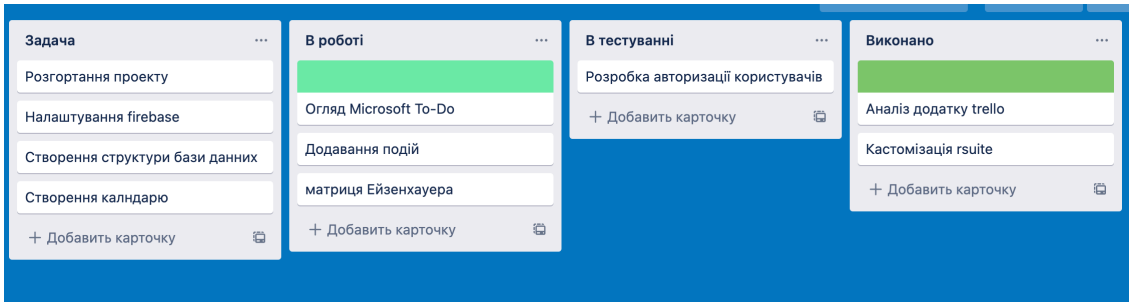


Рисунок 3.3 – Дошка з задачами спринту

3.2. Набір інструментів використаних для розробки

Середовищем для розробки проекту було обрано Gitpod (рис. 3.3). Gitpod являє собою веб сайт з вбудованим редактором коду Visual Studio Code що надає можливість писати код з будь якого пристрою маючи лише доступ до інтернету.

Він має вбудований термінал завдяки якому розробник має змогу встановлювати та використовувати будь які npm пакети або ж запускати JavaScript код в середовищі node.js.

Проект запускається на віртуальній машині створеного середовища та одразу відкриває порти до запущеного серверу на якому відображається створений веб додаток.

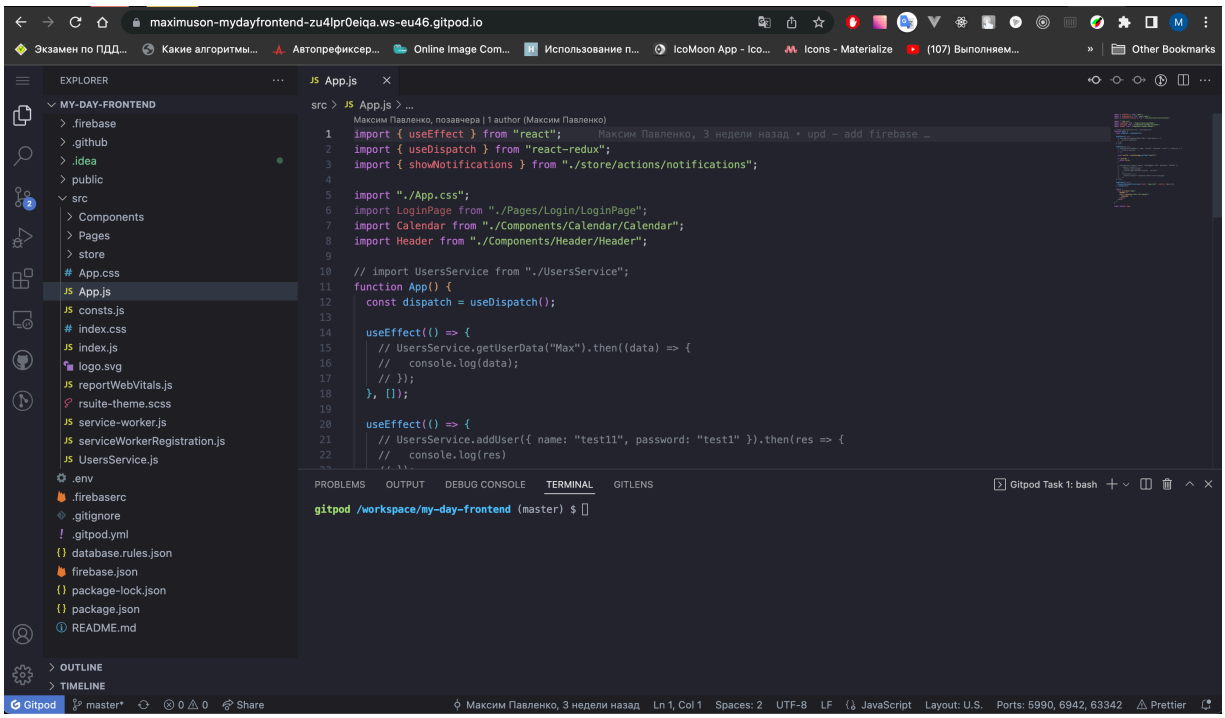


Рисунок 3.3 – Інтерфейс проекту в gitpod

Gitpod тісно пов’язаний з Github, щоб перейти до написання коду потрібно всього лише встановити плагін для браузеру google chrome та натиснути кнопку в репозиторії проекту (рис. 3.4)

Рисунок 3.4 – Перехід до gitpod в github

3.3. Функціонал клієнтської частини та його модулі

Клієнтська частина розподілена на компоненти що дає змогу використовувати одній й ті ж компоненти в різних місцях додатку.

Керування даними реалізовано за допомогою Redux, завдяки цьому доступ до отримання та керування задачами користувача може відбуватись з будь-якого компоненту. Це реалізовано з використанням хуків які містяться в бібліотеці react-redux (рис. 3.5).

```
const dispatch = useDispatch();

const date = useSelector( selector: (state : DefaultRootState ) => state.calendar.date);
const days = useSelector( selector: (state : DefaultRootState ) => state.calendar.days);

setCalendarDate();

useEffect( effect: () => {
  dispatch(setCalendarDate(new Date()));
  // eslint-disable-next-line
}, deps: []);
```

Рисунок 3.5 – Робота з даними за допомогою redux

При натисканні користувача на дату в календарі виконується функція що генерує нові дані для відображення календарю з задачами які беруться з бази даних (Рис. 3.6).

Метод створює дві дати. Дату початку відліку календарю яка базується на першому дні місяця від якого віднімається порядковий номер дня тижня першої дати місяця. Та кінцевій даті місяця до якої додаються дні тижня наступного місяця щоб строчка з днями тижням була заповненою до кінця.

```
const setDate = (state, action) => {
  const date = action.payload;
  // Calendar Start Day
  const d1 = new Date(date.getFullYear(), date.getMonth(), date: 1);
  d1.setDate(d1.getDate() - (d1.getDay() === 0 ? 7 : d1.getDay()));
  // Calendart End Day
  const d2 = new Date(date.getFullYear(), month: date.getMonth() + 1, date: 0);
  if (d2.getDay() !== 0) d2.setDate(d2.getDate() + (7 - d2.getDay()));

  const db = getDatabase();

  const days = [];
  do {
    // Obtain tasks
    d1.setDate(d1.getDate() + 1); // iterate
    days.push({
      date: new Date(d1.getTime()),
      tasks: db.filter((task) => sameDay(d1, task.date)),
    });
  } while (!sameDay(d1, d2));

  return updateObject(state, updatedProperties: {
    // Update state
    ...state,
    date: date,
    days: days,
  });
};
```

Рисунок 3.6. – Генерування календарю та внесення задач.

Головні модулі клієнтської частини:

- модуль з календарем та поставленими задачами;
- модуль статистики задач;
- модуль відображення задач за категоріями;
- модуль створення задачі;
- модуль авторизації.

3.4. Функціонал серверу

Головними задачами серверу є робота з користувачами, взаємодія з базою даних та хостинг веб додатку. Для забезпечення безпеки даних користувачі було встановлено правила на отримання та запис даних до бази (рис. 3.7)

Рисунок 3.7 – Встановлення правил доступа до таблиць бази

При авторизації користувачу надається токен, який встановлюється в заголовках до всіх запитів. При отриманні запиту на отримання даних сервер спочатку аналізує співпадіння токену користувача з об’єктом записаним в базі, ключем якого є id користувача.

Модуль з авторизацією надає змогу створювати користувачів за допомогою внутрішнього API сервісу. Розробник може обрати варіанти авторизації за допомогою пошти та пароля або ж з використанням інших сервісів. Всі створені користувачі відображаються в окремому вікні сервісу firebase (рис. 3.8).

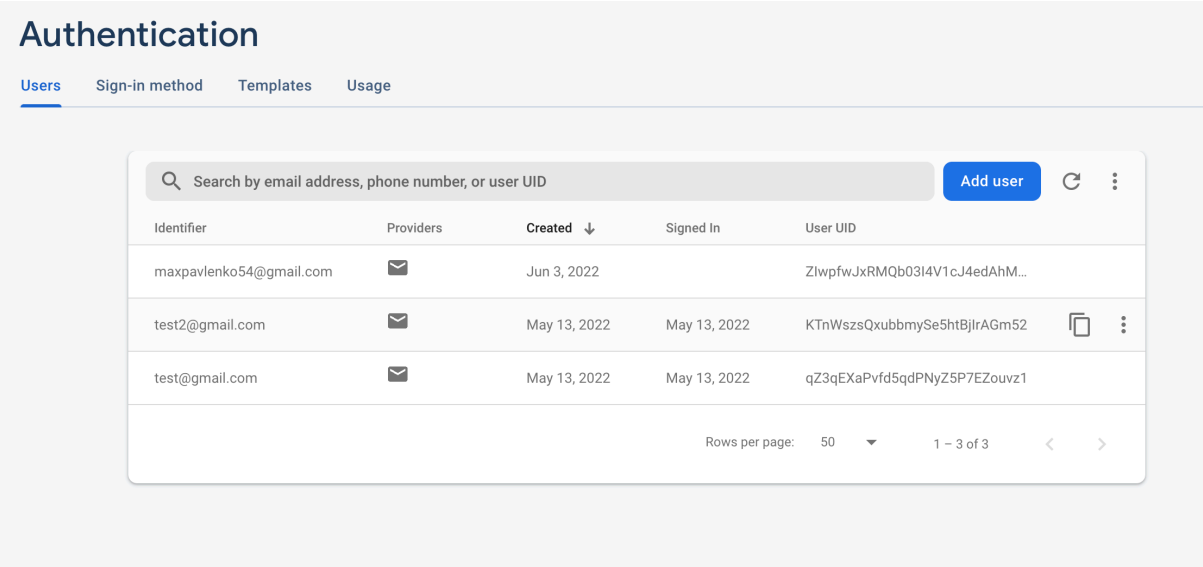


Рисунок 3.8 – Модуль керування користувачами

При реєстрації користувачу буде надіслано повідомлення на вказану пошту для валідації акаунту.

Для взаємодія з realtime database було створено клас UsersService який має статичні методи що надає змогу використовувати методи класу без створення його екземпляру (рис. 3.9).

Деплой проекту відбувається за допомогою firebase-cli. Щоб завантажити веб додаток потрібно запустити лише одну команду в терміналі “npm run build && firebase deploy”. Після запуску команди firebase завантажить нові файли з папки build та перезапустить сервер. Розробнику потрібно лише перевірити сайт перейшовши за посиланням яке буде виведено в консоль.

```
import axios from "axios";
import {FIREBASE_API_KEY} from "../consts"

class UsersService {
  static async getUserData(name :string  = "") {...}

  static async addUser({ name :string  = "", password :string  = "" }) {...}

  static async signUp({email, password}) {...}

  static async login({email, password}) {...}

  static async addTask(name, task) {...}
}

export default UsersService;
```

Рисунок 3.9 – Клас UsersService

4. ОПИС ФУНКЦІОНУВАННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1. Опис роботи додатку для самоменеджменту

Користувач повинен зареєструватися та увійти в систему. Після процесу реєстрацію користувачу буде надіслано лист на пошту для підтвердження акаунту. Далі користувач переходить на сторінку авторизації (рис.4.1.)

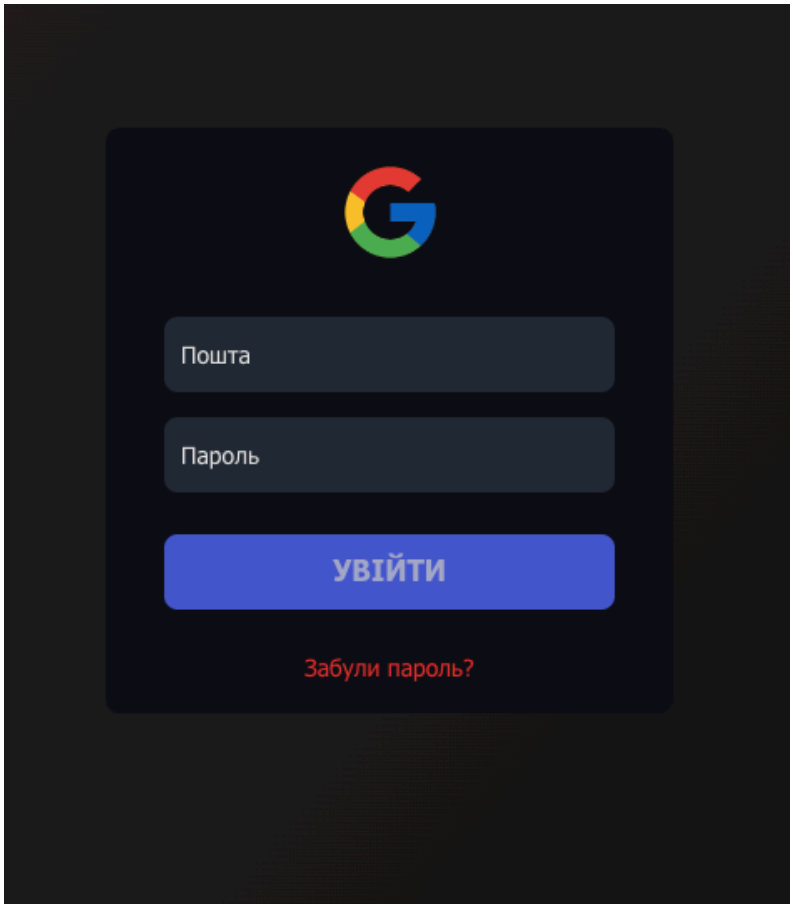


Рисунок 4.1 – Вікно авторизації

При натисканні на кнопку «забули пароль» та введенні пошти користувачу буде надіслано лист для встановлення нового паролю за допомогою сервісу firebase (рис. 4.2).

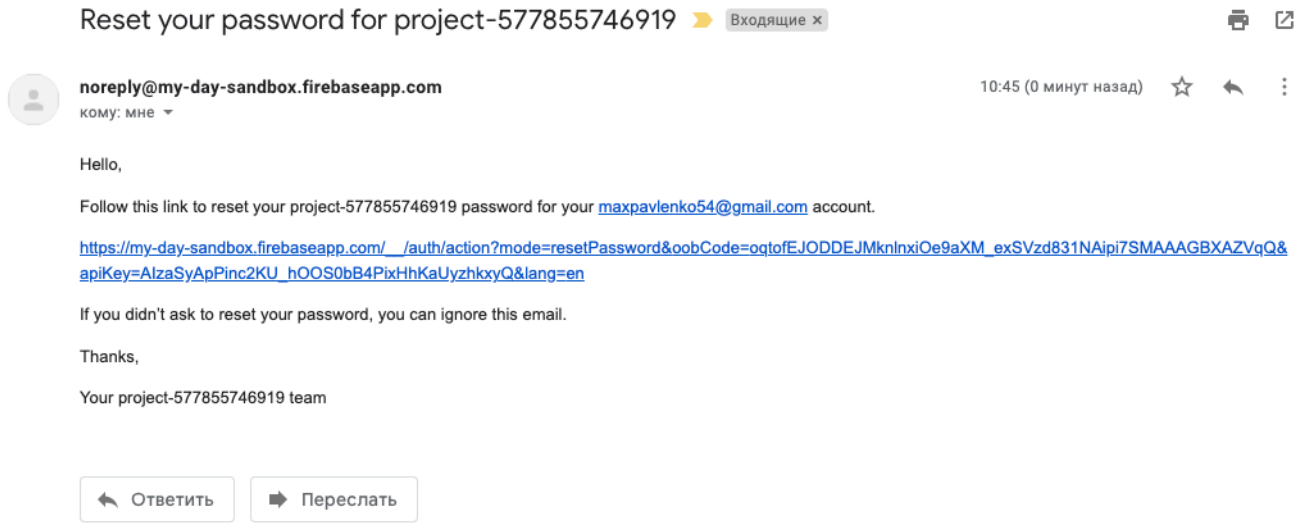


Рисунок 4.2 – лист для підтвердження зміни паролю

Після натискання на посилання в листі користувачу буде відкрито сторінку для встановлення нового паролю (рис. 4.3).

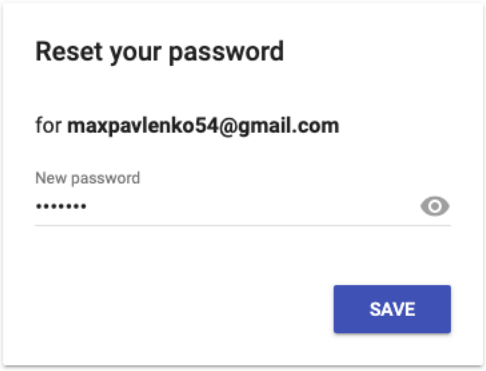


Рисунок 4.3 – Встановлення нового паролю.

Після успішної авторизації користувача потрапляє до сторінки з календарем на якому відображаються записано події з встановленими кінцевими датами задачі (рис. 4.4). В лівій частині знаходиться навігація завдяки якій можна переходити між сторінками.

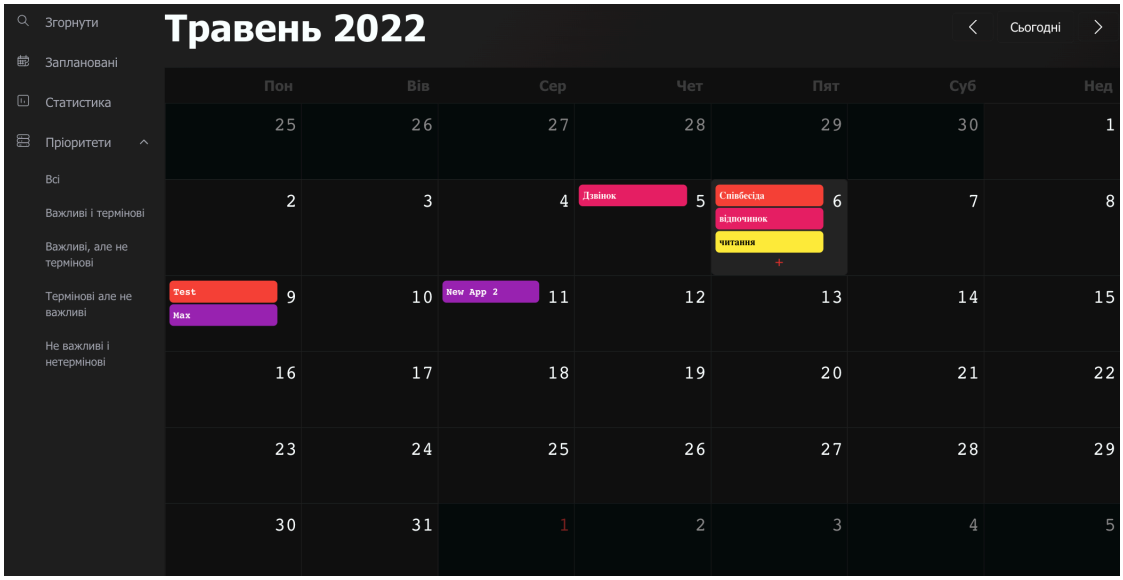


Рисунок 4.4 – Задачі користувача

При натисканні на кнопку додавання задачі в календарі користувачу відкривається модальне вікно в якому він може ввести назву задачі (Рис. 4.5), яка потім буде відображатися на компоненті календарю. Також є можливість додавати примітки з функцією форматування тексту. До задачі можна обрати колір заднього фону за допомогою компоненту ColorPicker.

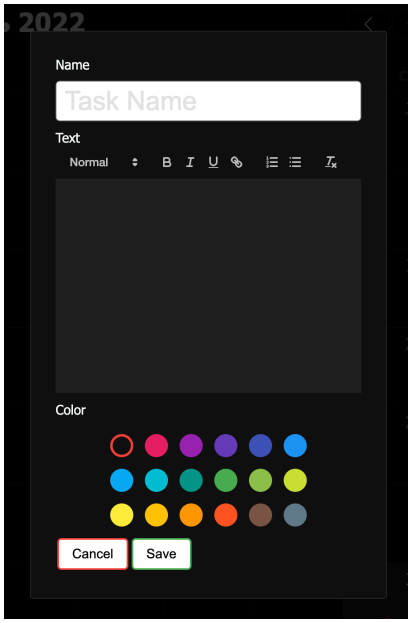


Рис. 4.5 – Компонент створення задачі

Однією з головних сторінок є сторінка з відображенням статистики (рис. 4.6). Користувач може переглянути кількість поставлених задач у вигляді відображення матриці Ейзенхауера, яка являє собою розподілення задач на координатній площина відсортованих за критеріями:

- важливі та термінові;
- важливі та не термінові;
- не важливі та термінові;
- не важливі та не термінові.

Кожній групі задач належить відповідний колір. Розміри блоку залежать пропорційно від кількості задач в блоку до відношення максимального блоку. Що надає зручність при аналізуванні груп з великою кількістю задач.

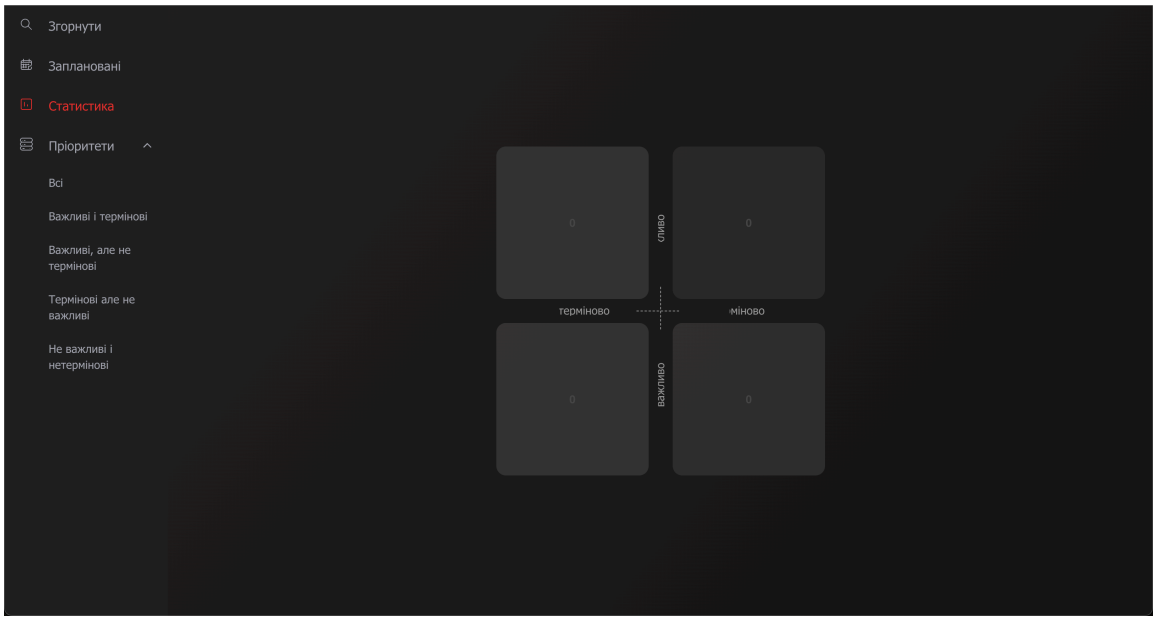


Рисунок 4.6 – Сторінка статистики задач

При переході на сторінку відбувається запит на отримання задач користувача. Після отримання данних створюється об’єкт з категоріями задач та кількістю задач відповідної категорії. Розмір максимального розміру блоку обмежений та складає 40% від висоти екрану користувача. Кожній категорії додано відповідний колір (рис. 4.7).

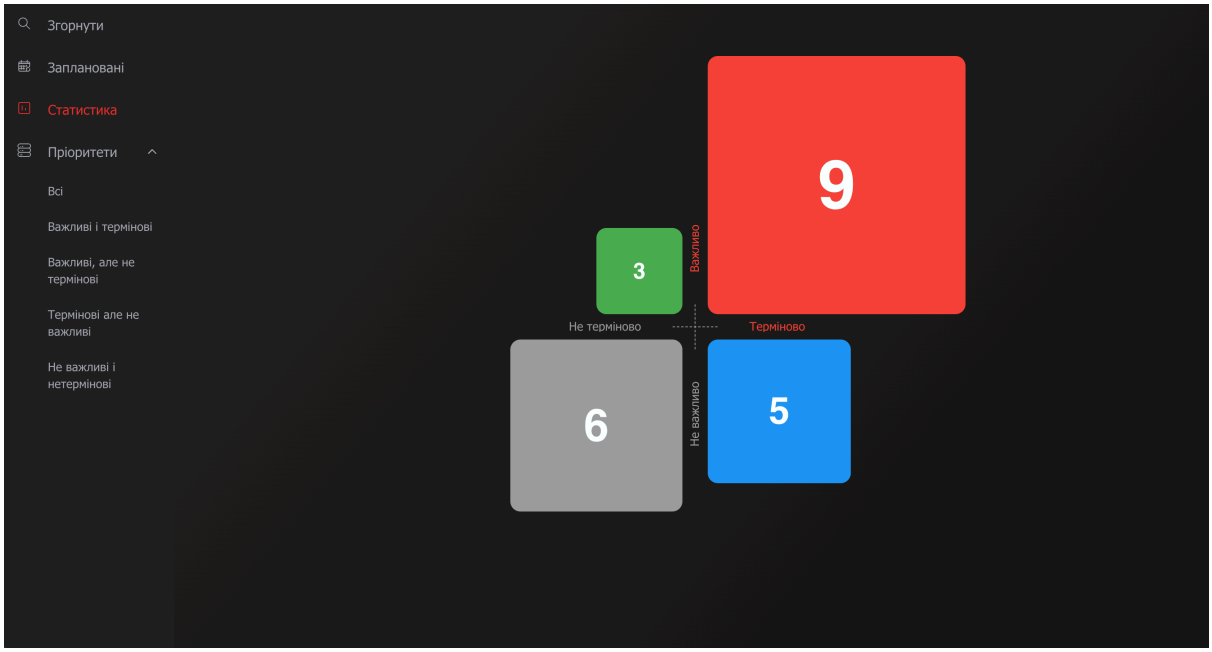


Рисунок 4.7 – Компонент статистики

4.2. Набір тестових сценаріїв для забезпечення якості продукту

Для тестування було обрано найпоширеніші бібліотеки Jest та ReactTestingLibrary. Jest - це бібліотека для виконання тестів на.JavaScript, яка надає змогу взаємодіяти з DOM браузеру через jsdom. Jest надає можливість тестувати react компоненти за допомогою знімків. Під час першого запуску тесту знімків він створює серійну версію відтвореного дерева React і зберігає цей знімок разом із тестом. Під час наступних запусків Jest може порівняти нещодавно відображений компонент із збереженим знімком, щоб попередити про будь-які відмінності.

React Testing Library — це набір допоміжних функцій, що дозволяє тестувати React-компоненти, не покладаючись на їхню внутрішню реалізацію.

Приклад написання тест кейсів (рис. 4.6)

```
test( name: 'handles click correctly', fn: async () => {
    render(<Checkbox />)
    const user = userEvent.setup()

    await user.click(screen.getByText( args: 'Check'))

    expect(screen.getByLabelText( args: 'Check')).toBeChecked()
})
```

Рисунок 4.6 – Приклад тест кейсу

Написання unit тестів допомагає уникнути проблем при рефакторингу коду або ж при створенні нового функціоналу. Ідеальним варіантом вважається покриття не менше ніж 80% коду. При тестуванні додатку було передбачено наступні сценарії:

- вхід користувача з некоректними даними;
- вхід користувача з коректними даними;
- створення задачі без назви;
- створення задачі з довгою назвою;
- створення багатьох задач на одну дату.

ВИСНОВКИ

Представлена робота спрямована на проектування та реалізацію додатку для покращення процесу створення особистого планування.

Обґрунтовано актуальність розробленого додатку, досліджено проблеми та потреби користувачів при роботі з додатками для самоменеджменту. В ході дослідження було виявлено потрібність користувачів в плануванні на основі чіткої методології.

Розроблено serverless архітектуру додатку з використанням сервісу firebase, що забезпечує безпеку даних користувачів та надає змогу швидко оновлювати продукт, переглядати статистику за допомогою гугл аналітики.

Обрано інструменти для розробки додатку такі як React, Redux, Firebase. Обґрунтовано перевагу та ефективність над іншими аналогами.

Розроблено клієнтську частину на основі бібліотеки React.js для побудови інтерфейсів користувача та бібліотеки Redux що надає змогу відокремити бізнес логіку від побудови інтерфейсів.

В ході розробки системи реалізовано додавання та редагування задач користувачем, планування їх з використанням компоненту календарю та перегляд статистики у вигляді групованої матриці поділеної на 4 категорії. Реалізовано додавання задачі до загального списку. Створено фільтри на основі матриці Ейзенхауера. Створено компонент для перегляду кількості важливих задач. Розроблено сповіщення користувача про надмірну кількість задач із розділу «А» матриці.

Описано програмні засоби та інструменти, які застосовувались в ході розробки програмного забезпечення. Середовищем для розробки було обрано веб сервіс Gitpod, оскільки він надає можливість взаємодіяти з віддаленим репозиторієм на github та створювати ізольовану віртуальну машину з розгорнутим проектом. Що надає змогу вести розробку з будь якого пристрою без встановлення потрібного ПЗ. Відстеження статусу виконання задач проекту здійснювалось за допомогою таск менеджера Trello

Написано тестові сценарії для перевірки коректності роботи компонентів. При тестуванні було обрано бібліотеки Jest та React Testing Library. Тестування процесу розгортання додатку відбувається за допомогою Firebase.

Створений продукт цілеспрямований на побудову особистого тайм менеджменту на основі методу розподілення задач по терміновості та важливості. Визначено що додаток може використовуватись для подолання проблеми з розподілом великої кількості задач. користувача

Результати дослідження бакалаврської роботи апробований на Всеукраїнській науково-технічних конференції «Діджиталізація науки як виклик сьогодення»

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Roy Sutton. Enyo: Up and Running: Build Native-Quality Cross-Platform JavaScript Apps. /- “O'Reilly Media”, 2015 – с. 87.

2. Google Chrome API Reference [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://developer.chrome.com/docs/extensions/reference/

3. Fielding Roy. Architectural Styles and the Design of Network-based Software Architectures./- 2000 – с. 180 – 185.

4. Mike Richardson, Ruby Leonard, Sam Amundsen.RESTful Web APIs./- “O'Reilly Media”, 2013 – с. 357 – 363.

5. Eric Elliott. Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Moderns JS Libraries. -/ “O’Reilly Media”, 2013 – с. 134– 177.

6. Any.do: [Веб-сайт]. URL: https://www.any.do/ (дата звернення: 22.04.2022)

7. Trello: [Веб-сайт]. URL: https://trello.com/ (дата звернення: 22.04.2022).

8. Laiza Krispin, Janette Gregory. Agile Testing: A Practical Guide for Testers and Agile Teams. /- 2010 – с. 215–247.

9. . Fielding Roy. Architectural Styles and the Design of Network-based Software Architectures./- 2000 – с. 180 – 185.

10. Канер Кем, Фолк Джек, Нгуен Енг Кек. Тестування програмного забезпечення. Фундаментальні концепції менеджменту бізнес-додатків./- 2001 – с. 344–375.

11. Introduction to REST API - RESTful Web Services [Електронний ресурс]. – Режим доступу до ресурсу: https://www.springboottutorial.com/introduction-to-rest-api

12. Бібліотека rsuitejs [Електронний ресурс] – Режим доступу до ресурсу: https://rsuitejs.com/.

13. PWA [Електронний ресурс] – Режим доступу до ресурсу: https://brainhub.eu/library/progressive-web-apps-advantages-disadvantages/.

14. JavaScript Design Patterns [Електронний ресурс] – Режим доступу до

ресурсу: https://www.dofactory.com/javascript/design-patterns.

15. Firebase console [Електронний ресурс] – Режим доступу до ресурсу: https://console.firebase.google.com/.

16. Матриця ейзенхауера [Електронний ресурс] – Режим доступу до ресурсу: http://www.management.com.ua/blog/3483.

17. Best Practices for Redux [Електронний ресурс] – Режим доступу до ресурсу: https://www.thisdot.co/blog/best-practices-for-redux.

18. Основи тайм менеджменту [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Time_management.

19. Огляд сутності самоменеджменту [Електронний ресурс]. – Режим доступу до ресурсу: http://www.km.lviv.ua/wp-content/uploads/2016/04/Samomenedzhment.pdf

20. Firebase as a service [Електронний ресурс] – Режим доступу до ресурсу: https://marmelab.com/blog/2019/10/23/feedback-on-firebase-in-project-start-up.html.

Додаток А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

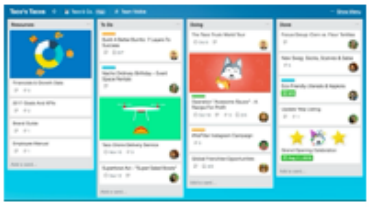
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-додатку для створення стратегії виконання та відстеження прогресу поставлених цілей на основі бібліотеки React.JS

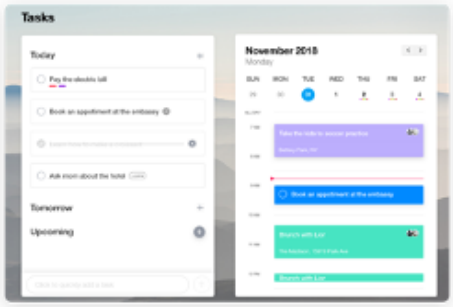
Виконав студент 4 курсу
групи ПД-44
Павленко Максим Вадимович
Керівник роботи
Золотухіна Оксана Анатоліївна
Доцент кафедри, кандидат технічних наук

Київ – 2022

Аналоги



Trello



Any.do

Evernote

Порівняння аналогів

Додаток	Переваги	Недоліки
Trello	Візуалізація даних у вигляді канбан дошки	Відсутня можливість сортування за категоріями
Any.do	Синхронізація між всіма пристроями, синхронізація задач з календарем, функція концентрування	Вартість, фільтрація лише за одним типом фільтру
Evernote	Великий набір функціоналу для створення детального опису, можливість створення власних рисунків	Обмежене використання безкоштовною версією, відсутній календар

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи спрощення процесу створення особистого структурованого плану задач за рахунок додавання показників якості планування на основі бібліотеки React.js

Об'єкт дослідження процес створення та аналізування особистого планування задач.

Предмет дослідження програмне забезпечення для створення особистого планування задач та контролю якості планування.

ТЕХНІЧНІ ЗАВДАННЯ

1. Аналіз потреб користувачів при роботі з додатками для самоменеджменту
2. Аналіз існуючих додатків з метою виявлення недоліків
3. Розробка архітектури клієнтської частини на основі бібліотеки React.js;
4. Розробка серверної частини на основі сервісу Firebase;
5. Розробка архітектури бази даних;
6. Реалізація головних модулів:
   - внесення в календар запланованої події;
   - додавання задачі до загального списку;
   - створення фільтрів на основі матриці Ейзенхауера;
   - перегляд кількості важливих задач;
   - сповіщення про надмірну кількість задач із розділу «А»;
   - автоматичне розподілення задачі по групам на основі обраних параметрів;

5

СТРУКТУРА МАТРИЦІ ЕЙЗЕНХАУЕРА

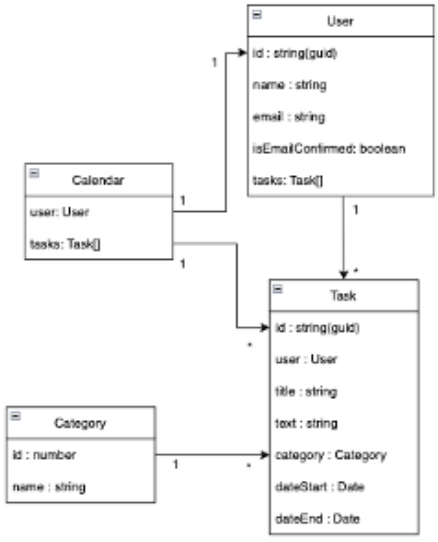
6

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

react redux firebase gitpod

7

Схема бази даних

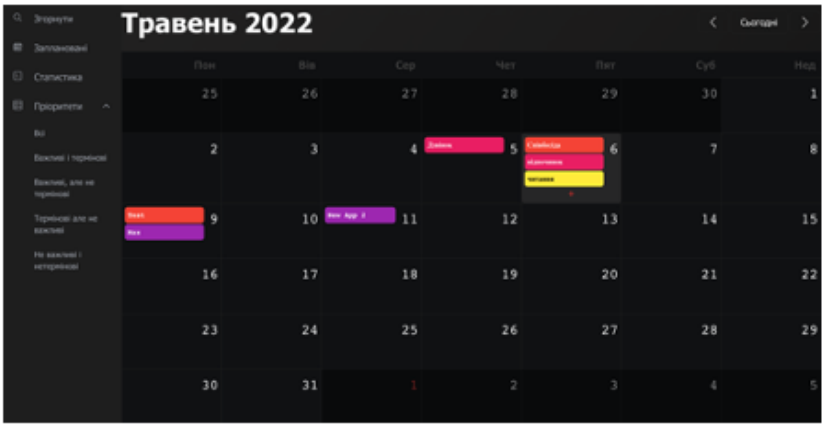


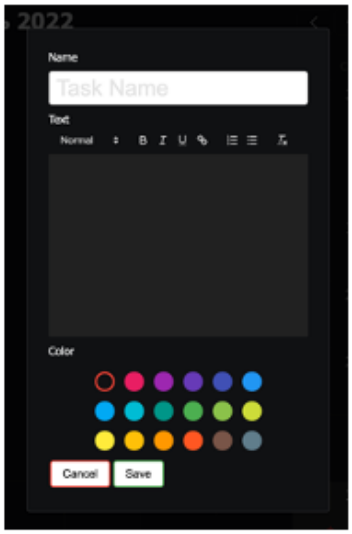
8

Діаграма прецедентів

9

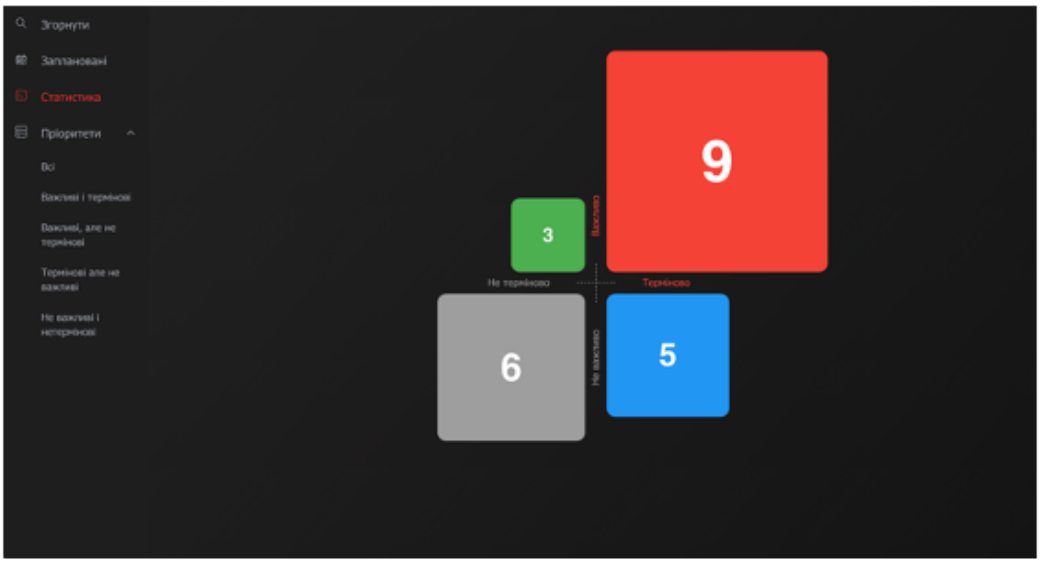
Екранні форми додатку





10

Екранні форми додатку



11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Павленко М.В. Безсерверні технології та firebase: III Міжнародної студентської наукової конференції «Діджиталізація науки як виклик сьогодення » (03.06.2022, м. Львів, Україна) С 179.

12

ВИСНОВКИ

1. Проаналізовано та виявлено основні потреби користувачів при роботі з додатками для самоменеджменту;
2. Проведено аналіз існуючих додатків з подальшим усуненням недоліків;
3. Розроблено архітектуру клієнтської частини на основі бібліотеки React.js;
4. Розроблено серверну частини на основі сервісу Firebase;
5. Розроблено архітектуру бази даних;
6. Реалізовано головні модулі:
   - внесення в календар запланованої події;
   - додавання задачі до загального списку;
   - створення фільтрів на основі матриці Ейзенхауера;
   - перегляд кількості важливих задач;
   - сповіщення про надмірну кількість задач із розділу «А»;
   - автоматичне розподілення задачі по групам на основі обраних параметрів;

ДЯКУЮ ЗА УВАГУ!