

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

РОЗРОБКА ОНЛАЙН-ПЛАНУВАЛЬНИКА ОСОБИСТОГО ЧАСУ

Текстова частина до курсової роботи
за спеціальністю 122, «Комп'ютерні науки»

Керівник курсової роботи

Канд.фіз.-мат. наук

Гречко А.В.

(прізвище та ініціали)

(підпис)

“ ____ ” _____ 2021 р.

Виконала студентка 3 курсу

Купчик Д.П.

(прізвище та ініціали)

“ ____ ” _____ 2021 р.

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЇВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

Кафедра мережних технологій

ЗАТВЕРДЖУЮ

Зав. Кафедри

Малашонок Геннадій Іванович

(підпис) _____
“ ____ ” _____ 2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студентки Купчик Дарини Павлівни факультету інформатики 3 курсу

Тема: Розробка онлайн-планувальника особистого часу

Зміст ТЧ до курсової роботи:

Календарний план

Вступ

Розділ 1: Аналіз предметної області. Постановка завдання курсової роботи

Розділ 2: Теоретичні відомості

Розділ 3: Опис реалізації програмного продукту

Висновки

Список використаної літератури

Додатки

Дата видачі “ ____ ” _____ 2020 р.

Керівник _____ Завдання отримала _____
(підпис) (підпис)

Календарний план виконання курсової роботи

Тема: Розробка онлайн-планувальника особистого часу

Календарний план виконання роботи:

№ п/п	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	Листопад 2020	
2.	Ознайомлення з існуючою інформацією по темі	Грудень 2020	
3.	Ознайомлення з існуючими системами аналогами	Січень 2021	
4.	Початок створення практичної частини	Лютий 2021	
5.	Початок написання теоретичної частини	Березень 2021	
6.	Аналіз практичної частини; її корегування	Квітень 2021	
7.	Подання проміжної версії практичної частини	Травень 2021	
8.	Остаточне завершення написання теоретичної частини роботи та розробки практичної частини; корегування	Травень 2021	
9.	Створення презентації	Травень 2021	
10.	Захист курсової роботи	Після 24 травня	

Студентка Купчик Д.П.

Керівник Гречко А.В.

“ _____ ” _____

ЗМІСТ

Перелік термінів та умовних позначень	5
Вступ	6
Розділ 1	7
1.1. Аналіз сучасного стану питання та обґрунтування теми	7
1.2. Огляд існуючих аналогів розробки	8
1.2.1 Todoist	8
1.2.2 TickTick	10
1.2.3 Notion	12
1.2.4 Висновок	13
1.3. Постановка завдання	13
Розділ 2	14
2.1 Теоретичні відомості	14
2.1.1 Основні види трекінгу часу	14
Розділ 3	15
3.1 Аналіз технічного завдання	15
3.1.1 Основні відомості	15
3.1.2 Схеми-інтерфейсу	16
3.1.3 User flows	21
3.2 Обґрунтування вибору засобів розробки	24
3.2.1 Front-end	24
3.2.2 Back-end та бази даних	24
3.3 Опис розробки програми	26
3.4 Створення об'єктів і розробка головної програми	36
3.5 Опис файлів даних та інтерфейсу програми	36
3.6 Тестування програми і результати її виконання	44
Висновок	47
Список використаної літератури	48
Додатки	50

Перелік термінів та умовних позначень

UI (від англ. User Interface - «призначений для користувача інтерфейс») – Інтерфейс являє собою графічну структуру програми. Він складається з кнопок, натиснених користувачами, текстів, які вони читають, зображень, полів введення тексту і всіх інших елементів, з якими взаємодіє користувач.

UX (від англ. User Experience - «користувацький досвід») - призначений для користувача досвід визначається тим, як користувачі взаємодіють з додатком/сервісом.

Трекер (від англ. track – «відстеження») - така таблиця, схема, яка дозволяє нам відстежувати певні показники, події в нашому житті і те, як часто вони відбуваються.

Трекінг – процес відстеження.

Todo list – список із певних справ, які потрібно виконати.

Кастомізація (від англ. customize – «налаштування») - процес адаптації та налаштування продукту під окрему аудиторію, об'єднану певними особливостями.

Диджиталізувати - переведення інформації у цифрову форму.

Лендінг – це цільова сторінка, яка спонукає відвідувача виконати певну дію.

User flow - перехід користувачів від одного сценарію взаємодії з інтерфейсом до іншого, очікуваний алгоритм дій користувача.

Хук – нова можливість в бібліотеці React, яка дозволяє використовувати стан та інші можливості React без написання класу.

Вступ

Все життя нас переслідують завдання різного типу та різного рівня складності, які певним чином впливають на наше майбутнє. Неважливо, чи маєте Ви справу з діловими завданнями або просто ставите особисті цілі, - все це вимагає грамотного управління, адже ефективне управління часом дає переваги як, наприклад, з'являється більше вільного часу, та покращуються всі аспекти життя, будь то на роботі чи вдома.

Helru – зручний у використанні та розумінні веб-застосунок який дозволяє будь-якій людині доступно та швидко відслідковувати всі свої завдання.

Метою даної курсової роботи – допомогти людям ефективно керувати своїм часом через планування своїх справ.

Завданням даної курсової роботи є розробка такого застосування, яке задовольнятиме основні потреби користувачів у відстеженні свого часу та планування справ і буде простим та зручним у використанні.

Предметом дослідження є розробка онлайн застосунку, який допоможе людям ефективніше контролювати особистий час.

Об'єкт дослідження - існуючі на ринку системи-аналоги, підходи, методи та засоби розробки веб-сервісів, призначених для боротьби з неефективним використанням особистого часу.

Практичне значення одержаних результатів дослідження полягає в тому, що аналіз наявних сервісів дозволяє врахувати їхні переваги та недоліки і на основі цих даних розробити власний сервіс, що буде поєднувати в собі сильні сторони наявних платформ.

Для створення Helru було використано:

- Для побудови прототипу сайту на початковому етапі: Figma - онлайн-сервіс розробки інтерфейсів та прототипування.

- Для клієнтської частини: мова програмування JavaScript з використанням бібліотеки React, бібліотеки Redux та стилів CSS.
- Для серверної частини: Cloud Firestore - гнучка, масштабуєма хмарна база даних від Firebase і Google Cloud Platform для веб-застосунків, мобільних платформ, і серверних додатків.
- Для встановлення та структурування пакетів: NPM - Node Package Manager.
- Для контролю та написання коду: середовище JetBrains WebStorm.

Робота складається зі вступу, трьох основних розділів, кожен з яких містить підрозділи, висновків, списку використаних джерел та додатків.

У першому розділі проаналізовано тему як актуальне питання для дослідження, розглянуто аналоги розробки, які наразі перемагають на ринку, зроблені певні висновки та сформульовано постановку задачі.

Другий розділ висвітлює теоретичні відомості про предметну область.

У третьому розділі описується реалізація проекту. Визначені всі необхідні вимоги до даних, які представлені у застосунку, схематично зображені інтерфейси, сценарії користувачі, наведені деталі процесу розробки, обґрунтовано вибір використаних технологій розробки та продемонстровано результати.

РОЗДІЛ 1

1.1. Аналіз сучасного стану питання та обґрунтування теми

Спираючись на статтю від Development Academy [1] на 18 лютого 2021 року, тільки 18% всіх людей використовували якусь систему/додаток для контролю свого часу. Це може бути спричинено багатьма факторами, за словами компанії

ProductIO [2] одними з них є факт того, що планування своїх справ є дуже довгим та занадто складним.

Не зважаючи на це, останній рік багато чого змінив, адже через карантин та роботу з дому у людей з'явилося більше вільного часу, що означає більше часу для нових справ. Саме тому тема правильного тайм-менеджменту та ефективного планування своїх справ з кожним днем стає все важливішою та актуальнішою, адже, як зазначено у статтях [3, 4], ефективне планування покращує як фізичне так і ментальне здоров'я людини.

Отже, як наслідок усіх вищезазначених факторів, сервіси для планування справ, так звані task management apps, починають дуже стрімко розвиватись та набирати популярності, оскільки люди все більше і більше хочуть бути продуктивнішими та як можна швидше досягати поставлених ними цілей.

1.2. Огляд існуючих аналогів розробки

На сучасному ринку програмного забезпечення доступні сотні застосунків для трекінгу персональних справ та часу, серед яких у багатьох статистиках лідерами можна виділити Todoist, TickTick та Notion, підтвердження чого є статистика College Info Geek [5], ресурсу, який націлений на допомогу щодо ефективного навчання та продуктивності.

1.2.1 Todoist

Todoist [6] - таскменеджер, веб-сервіс і набір програмного забезпечення для управління проектами та керування завданнями. Це не найпотужніший трекер справ, хоч і збалансовує потужність простотою, і робить це, працюючи практично на кожній існуючій платформі роблячи - імовірно, саме тому Todoist зараз є одним із найпопулярніших списків справ, у якому понад 10 мільйонів користувачів.

Todoist досить гнучкий, щоб адаптуватися до більшості робочих процесів, але не настільки складний, щоб перевантажити. Загалом, це чудовий перший додаток зі

списком справ, який можна спробувати, особливо якщо Ви не знаєте, з чого почати.

Основний функціонал:

- Створення завдань та призначення дедлайнів.
- Drag & Drop.
- Управління завданнями.
- Пріоритетність.
- Списки справ.
- Залежності.
- Календар.
- Повторювані завдання.

Приклад інтерфейсу головної сторінки можна побачити на Рисунку 1.

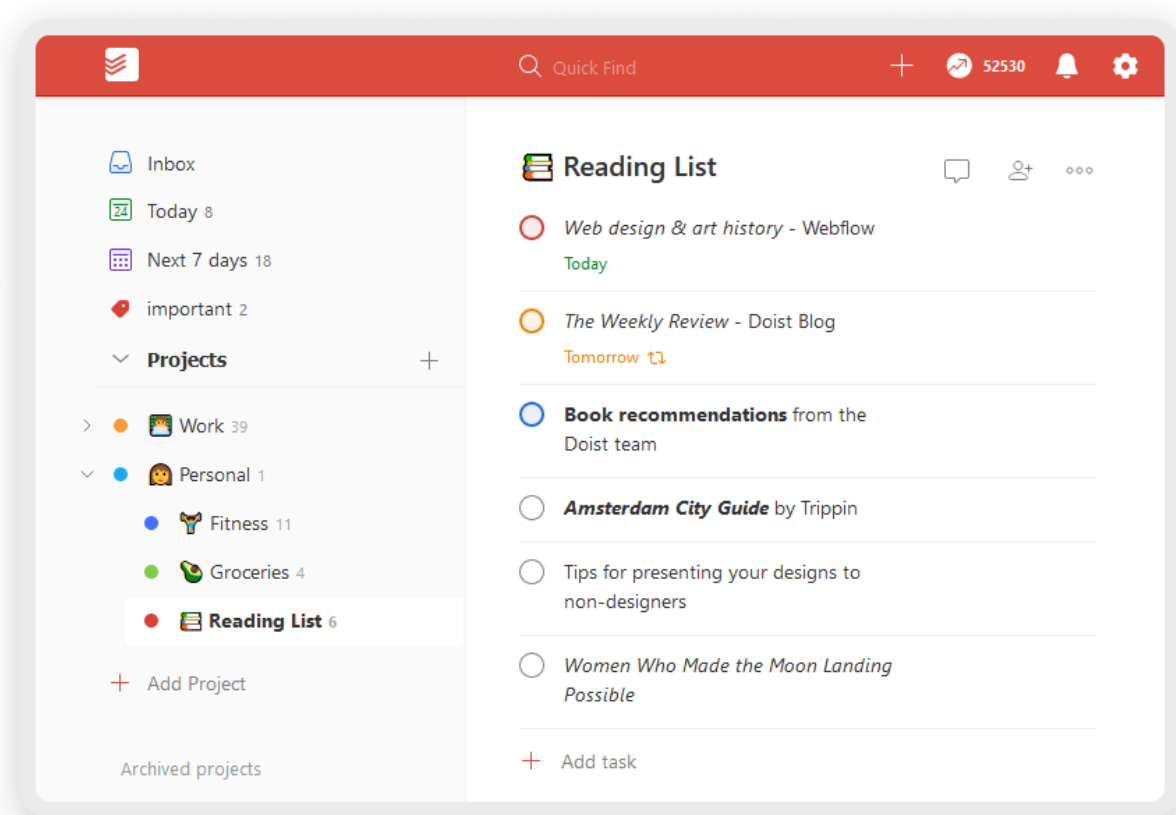


Рис. 1. Todoist – головна сторінка користувача [7]

Нижче наведена таблиця переваг та недоліків даного застосування (Таблиця 1):

№	Переваги	Недоліки
1	Гнучка для будь-якої системи продуктивності	Немає можливості переглянути виконані справи
2	Чудова підтримка	Іноді може працювати з перебоями
3	Підтримка між платформами з легкою та надійною синхронізацією	Не дуже зрозумілий інтерфейс з фільтрами та пошуком
4	Працює в автономному режимі	У безкоштовній версії не весь функціонал доступний.
5	Відмінні функції, такі як введення на природній мові та звіти про продуктивність	-

Таблиця. 1. Todoist – переваги та недоліки

1.2.2 TickTick

TickTick [8] - це швидкозростаючий додаток зі списком справ, який пропонує широкий спектр функцій майже на кожній платформі, яку ви можете собі уявити. Додавання завдань відбувається швидко завдяки обробці природної мови. Завдання можна організувати за допомогою списків, тегів та термінів виконання, і навіть є можливість додавати підзадачі до будь-якого завдання.

Основний функціонал:

- Створення завдань та призначення дедлайнів.
- Шаблони.
- Календар.
- Гнучке налаштування дати.
- Сортування за будь-якими критеріями.
- Пріорітезація.

Приклад інтерфейсу головної сторінки можна побачити на Рисунку 2.

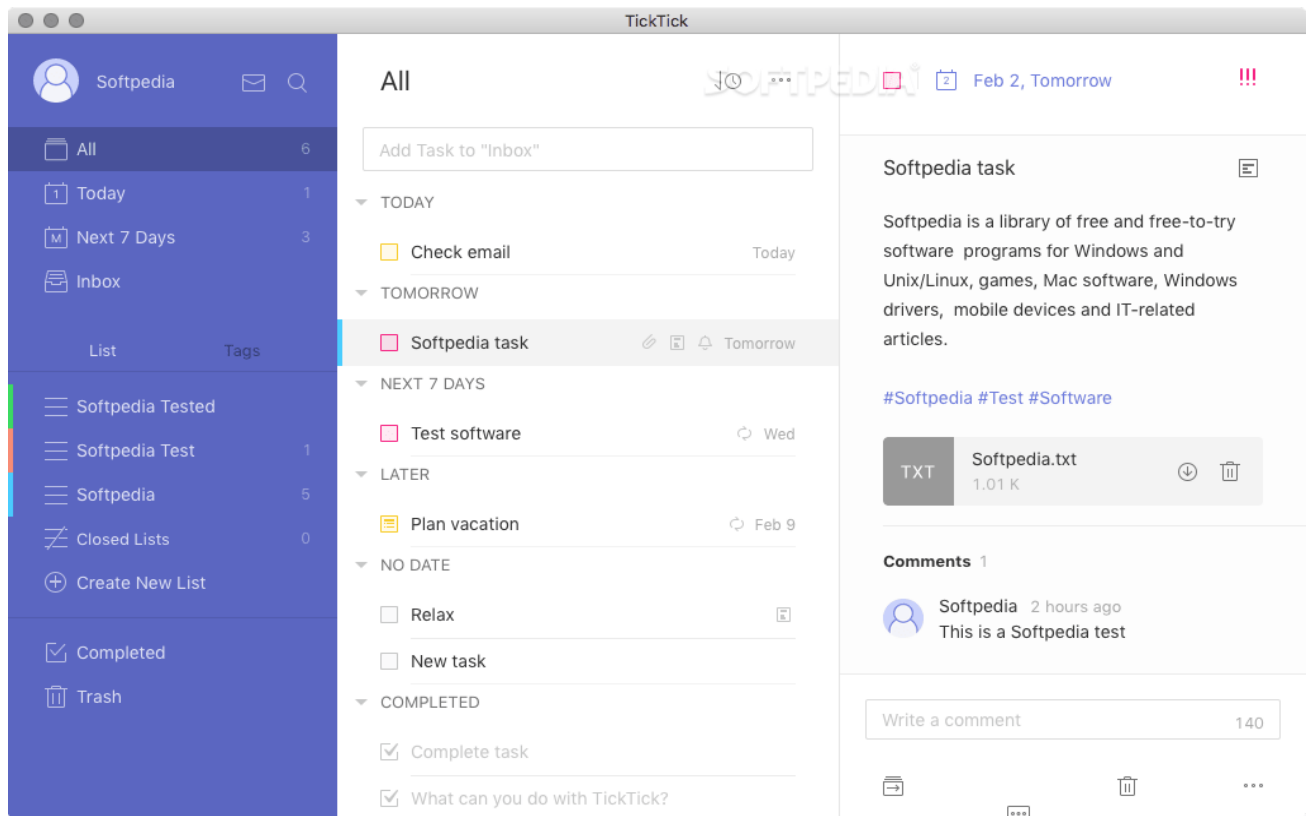


Рис. 2. TickTick – головна сторінка користувача [9]

Нижче наведена таблиця переваг та недоліків даного застосування (Таблиця 2):

№	Переваги	Недоліки
1	Є вбудований таймер Pomodoro	Немає функції нагадувань
2	Існує інтеграція з різними сторонніми календарями	Трохи складний UI
3	Є вбудований інструмент відстеження звичок	

Таблиця. 2. Ticktick – переваги та недоліки

1.2.3 Notion

Notion [10] - відносно новий додаток для створення нотаток та популярний серед студентів та підлітків. Він рахується гібридним додатком для створення нотаток та менеджер завдань, але не має добре організованої структури для чіткого та простого вирішення цих двох потреб.

Основний функціонал:

- Створення завдань та призначення дедлайнів.
- Робота з документами.
- Календар.
- Створення баз знань.
- Управління проектами.

Приклад інтерфейсу головної сторінки можна побачити на Рисунку 3.

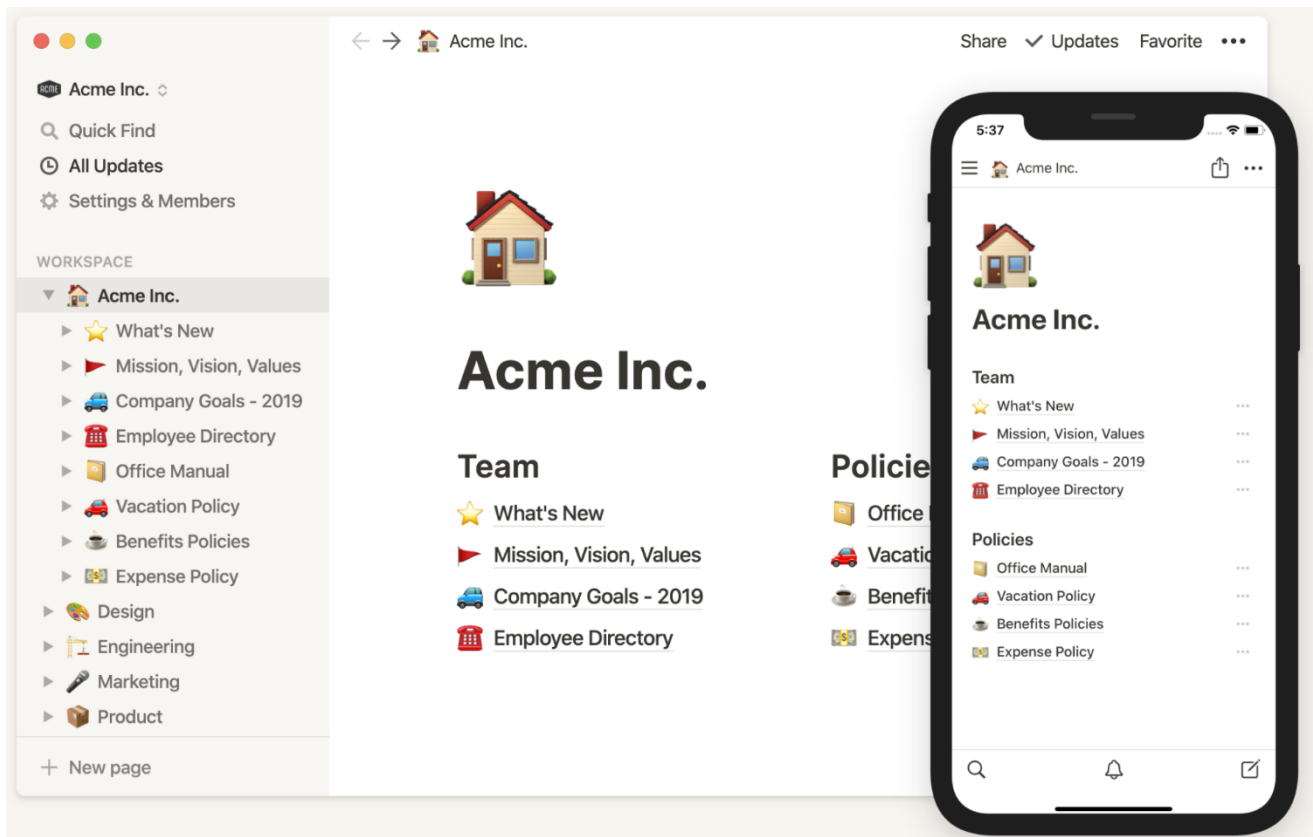


Рис. 3. Notion – головна сторінка користувача [11]

Нижче наведена таблиця переваг та недоліків даного застосування (Таблиця 3):

№	Переваги	Недоліки
1	Чистий та зручний UI	Налаштування вимагає часу, особливо для співпраці в команді
2	Єдина платформа з декількома інструментами, що економить користувачам дорогий час	Пошук та навігація потребують вдосконалення
3	Високий рівень кастомізації	Форматування тексту погано працює при використанні копіювання / вставлення
4	Ієрархічна структура	Треба певний час щоб звикнути та зрозуміти як знайти те, що тобі треба на сайті
5	Правильно фіксує синтаксис коду	-

Таблиця. 3. Notion – переваги та недоліки

1.2.4 Висновок

Проаналізувавши існуючі на ринку веб-сервіси можна побачити, що необхідними критеріями якісного планувальника є можливості створювати та редагувати завдання/справи, наявність календарю та трекінг дедлайнів.

Також, достатньо часто користувачів відлякує не дуже продуманий та іноді складний інтерфейс програми.

1.3 Постановка завдання

Завданням даної курсової роботи є розробка онлайн-застосунку для планування особистого часу, що допоможе користувачам структурувати їхні справи, мати можливість переглянути всі свої завдання в календарі та бачити свою статистику.

Таким чином можна сформулювати перелік необхідних функцій для повноцінної роботи системи:

- Авторизація та реєстрація в системі.
- Перегляд своєї статистики по справах (скільки всього спав, скільки вже виконано і скільки залишилось виконати);
- Створення своїх подій/справ.
- Створення своїх категорій подій.
- Редагування категорій подій.
- Додавання до календаря події.
- Перегляд списку справ на обраний день/тиждень/період.
- Відмітки для зроблених справ.
- Відмітки hot для певних подій, які означатимуть, що виконання даної справи має найвищий пріоритет.
- Перегляд нагадування про необхідність виконання певної справи.

РОЗДІЛ 2

2.1 Теоретичні відомості

2.1.1 Основні види трекінгу часу

Task manager app – застосунок або сервіс, який використовується окремою особою, командою або організацією для ефективного виконання проєктів або персональних справ шляхом організації та встановлення пріоритетів відповідних завдань. Існують різні види та реалізації цих інструментів управління завданнями та часом.

Перший та найочевидніший спосіб це запам'ятовування та тримання всіх своїх справ у себе в голові. Хоч це і найшвидший спосіб, але точно не найнадійніший, будь-яка важлива справа може в найнеочікуваніший момент вилетіти з голови та зіпсувати всі плани.

Другий варіант – паперовий метод. Безсумнівно кожний з нас хоча б раз в житті виписував всі свої завдання на папері у так званий todo list, щоб нічого не

забути і мати повний список справ завжди перед очима, але не завжди даний метод допомагав, адже листочок міг просто загубитись, або задля редагування справи треба було все перекреслювати і писати наново, створюючи безлад на робочому аркуші.

Третій і остаточний метод трекінгу – онлайн трекери, спеціальні сервіси або інструменти, які допомагають диджиталізувати перелік всіх своїх справ. Такі програми відслідковування часу та завдань дозволяють легко відстежувати час, контролювати виконані справи, завдання, які залишились, та структурувати всю інформацію задля ефективного розподілу свого часу та сил.

РОЗДІЛ 3

3.1 Аналіз технічного завдання

3.1.1 Основні відомості

Як уже було зазначено вище, завданням даної курсової роботи є розробка веб-застосування, яке дозволить будь-якій людині доступно та швидко відслідковувати всі свої завдання/події.

Призначення системи – спрощення контролю виконання, відслідковування та планування особистих справ.

В основі підходу до планування особистого часу лежить поняття події. Подія – це певна сутність, що визначає докладні характеристики деякої справи такі як назва, категорія, час та день виконання, можливий (необов'язковий) опис, умови тощо.

Приклади категорій справ: робота по дому, навчання, розваги тощо.

Приклади справ: похід в магазин, зміна постільної білизни, виконання лабораторної роботи, прогулянка з друзями в парку, побачення тощо.

В застосунку наявна одна група користувачів – звичайні юзери. Технічні вимоги користувачів до функціональності системи були перелічені у постановці задачі.

На етапі дослідження та постановки задачі було розроблено прототип сайту в середовищі Figma [12] – певні схеми-інтерфейсу задля більш детальної візуалізації сайту. За допомогою цих схем нижче буде показано більш детальну інформацію перебігу користувача по сайту.

3.1.2 Схеми-інтерфейсу

Коли користувач вперше потрапляє на сайт, він потрапляє на головну сторінку – лендінг сайту (Рис. 4). Як показано на схемі, справа розташована ілюстрація-вітання, а справа певний ознайомчий текст з кнопками входу та реєстрації.

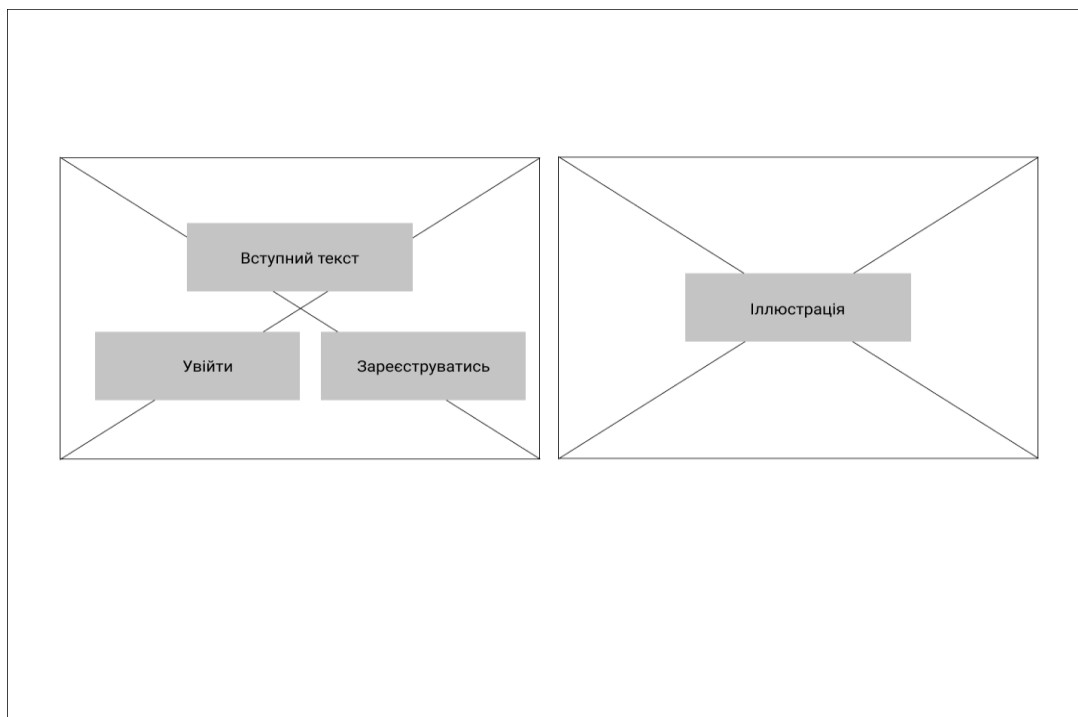
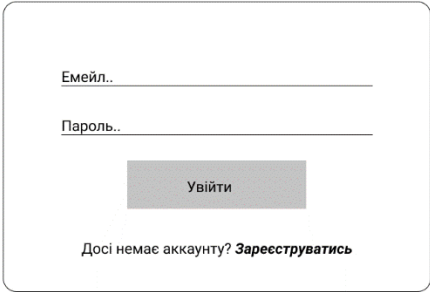


Рис. 4. Головна сторінка застосунку

Натиснувши на одну з кнопок користувач опиниться на сторінці авторизації (Рис.5). Якщо користувач не зареєстрований він може перейти на сторінку реєстрації натиснувши на «Зареєструватись».



Емейл.. _____

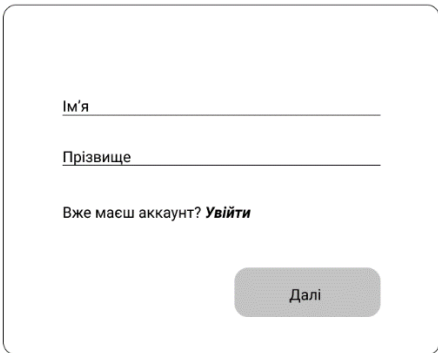
Пароль.. _____

Увійти

Досі немає аккаунту? [Зареєструватись](#)

Рис. 5. Сторінка входу

Перейшовши на сторінку реєстрації, користувачу треба буде пройти степпер, який матиме три частини: 1 - заповнити ім'я та прізвище (Рис.6), 2 – вибрати шаблонні категорії (Рис. 7), 3 – заповнити електронну пошту та пароль (Рис. 8).



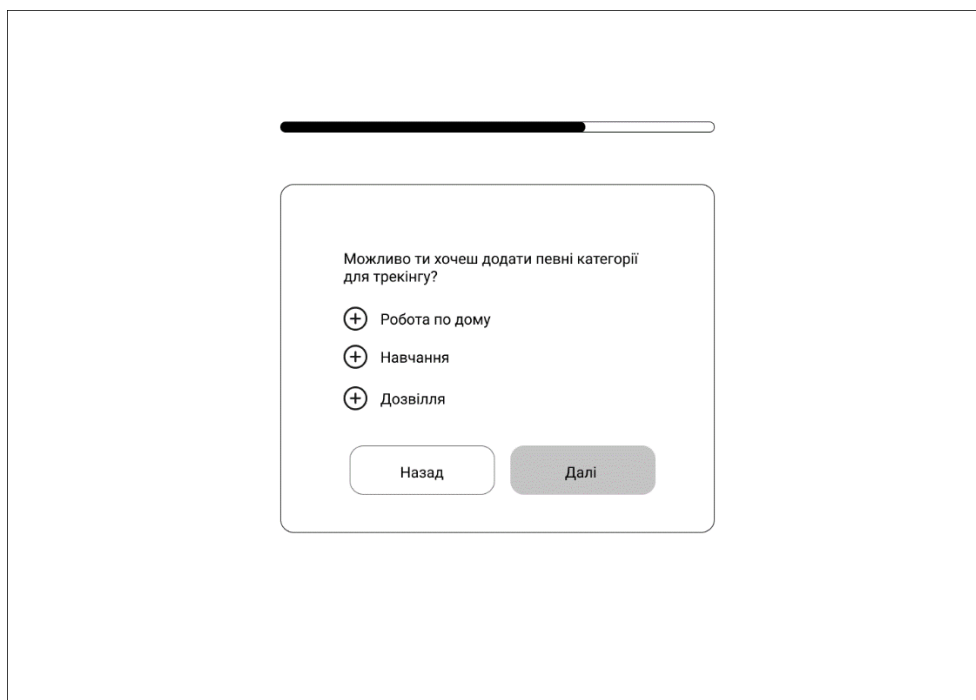
Ім'я _____

Прізвище _____

Вже маєш аккаунт? [Увійти](#)

Далі

Рис. 6. Сторінка реєстрації – крок 1



A registration screen with a white background and a thin black border. At the top, there is a thick black horizontal bar. Below it, a rounded rectangle contains the text "Можливо ти хочеш додати певні категорії для трекінгу?". Underneath this text are three options, each preceded by a plus sign in a circle: "Робота по дому", "Навчання", and "Дозвілля". At the bottom of the rounded rectangle are two buttons: "Назад" (light gray) and "Далі" (dark gray).

Рис. 7. Сторінка реєстрації – крок 2



A registration screen with a white background and a thin black border. At the top, there is a thick black horizontal bar. Below it, a rounded rectangle contains two input fields: "Електронна пошта" and "Пароль". At the bottom of the rounded rectangle are two buttons: "Назад" (light gray) and "Завершити" (dark gray).

Рис. 8. Сторінка реєстрації – крок 3

Після заповнення всіх необхідних полей у вході або реєстрації, користувача перенаправлює на головну сторінку самого застосунку (Рис. 9). Зліва розташоване головне меню сайту у якому є іконки сторінки юзера, головної сторінки, календаря та сторінки на додавання категорії або події.

Справа розташована основна інформація: нагадування, справи на сьогодні і шаблонні блоки з інформацією.



Рис. 9. Головна сторінка авторизованого юзера

Натиснувши на іконку профіля юзера, користувач може побачити інформацію про свій аккаунт та свою статистику (Рис. 10).



Рис. 10. Профіль юзера

Натиснувши на іконку календаря, користувач може побачити календар зі всіма своїми запланованими справами (Рис. 11).

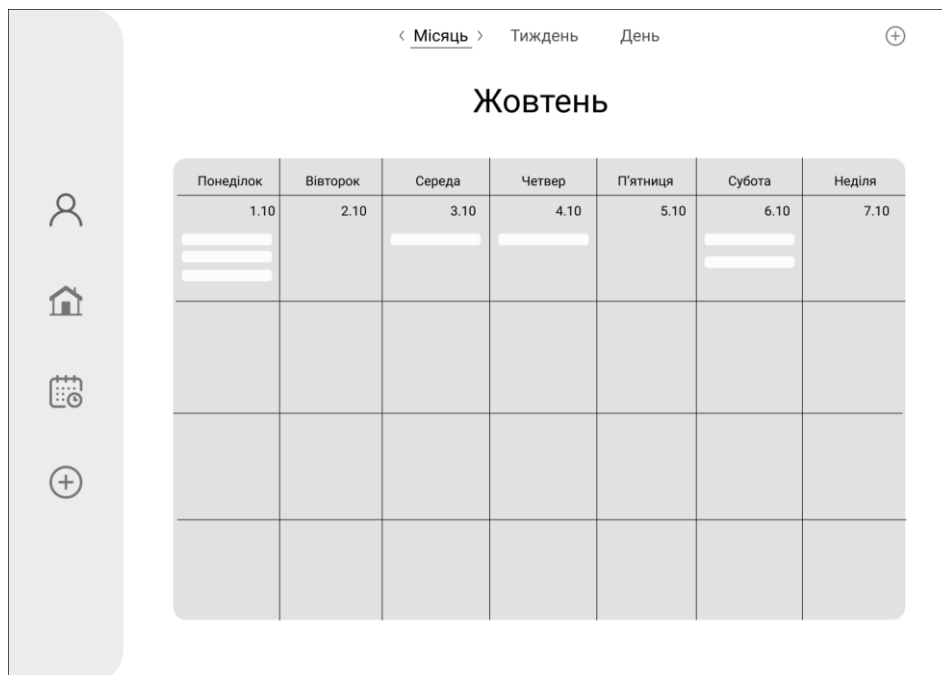


Рис. 11. Календар

Натиснувши на іконку додати, користувач опиниться на сторінці з відповідною формою додавання категорії чи події (Рис. 12).

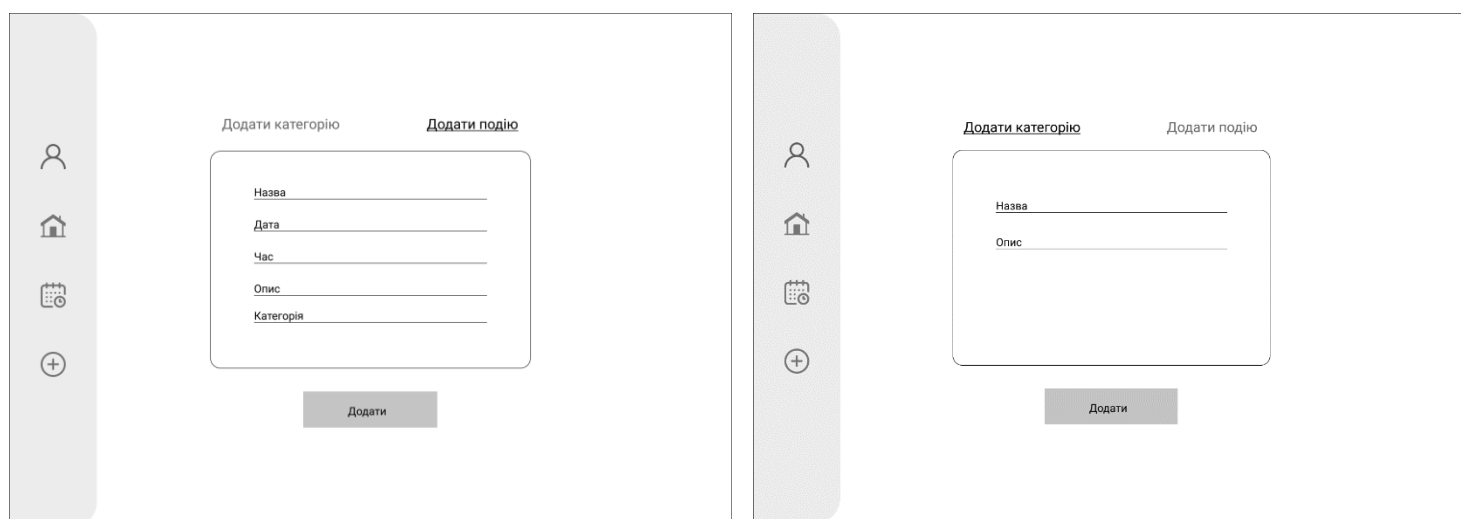


Рис. 12. Сторінка створення нової категорії/події

3.1.3 User flows

Розробивши інтерфейс програми можна чітко виділити сценарії користувачів, які надалі допоможуть розділити програму на модулі. Усі сценарії показані як блок-схеми.

Сценарій авторизації можна побачити на Рисунку 13.

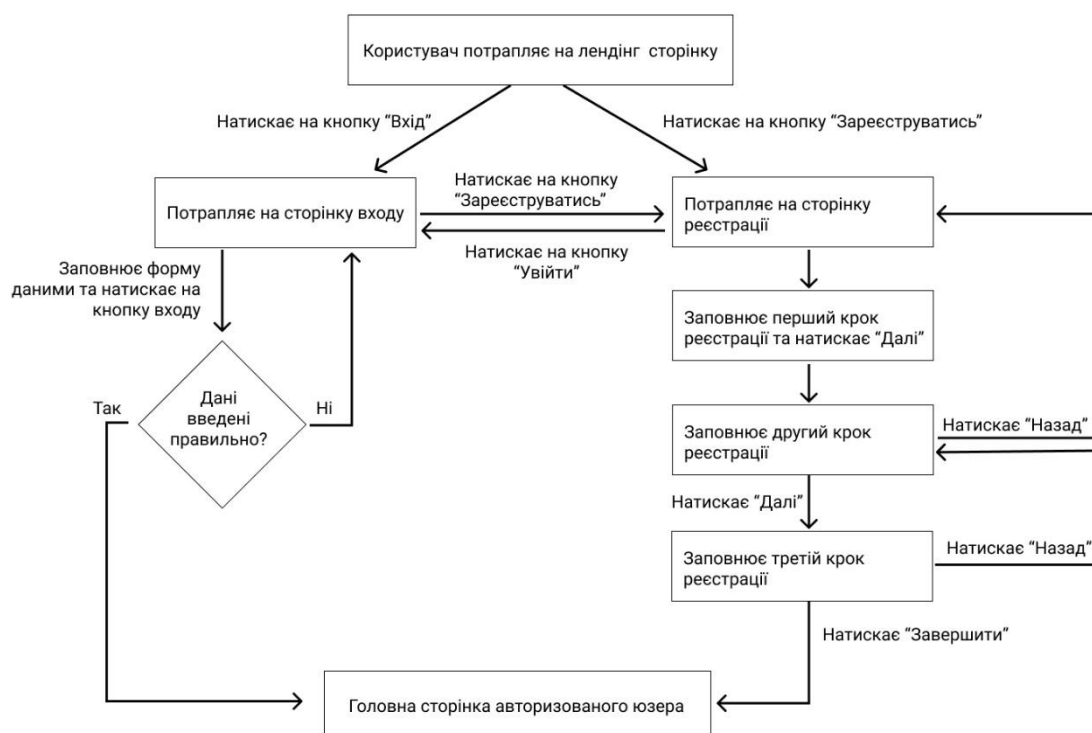


Рис. 13. Сценарій користувача – авторизація

Після того як користувач авторизувався він має чотири доступні сторінки, на які може потрапити, кожна з яких дозволяє користувачу зробити певні дії.

Сценарій головної сторінки можна побачити на Рисунку 14.

Сценарій сторінки профіля юзера можна побачити на Рисунку 15.

Сценарій сторінки календаря можна побачити на Рисунку 16.

Сценарій сторінки додавання можна побачити на Рисунку 17.

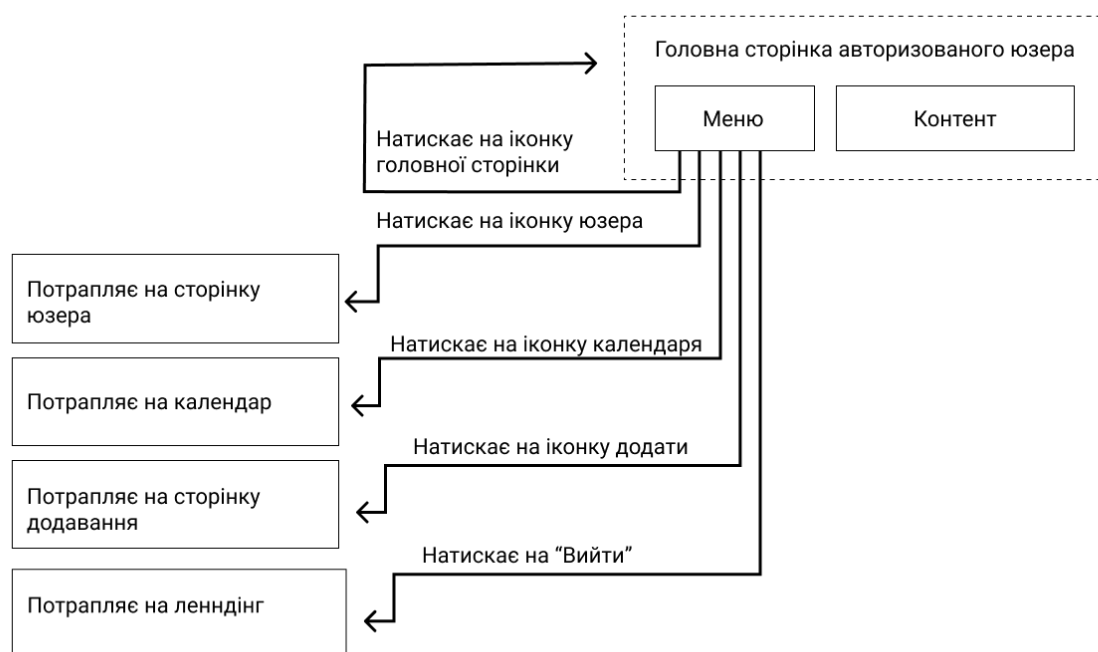


Рис. 14. Сценарій користувача – головна сторінка

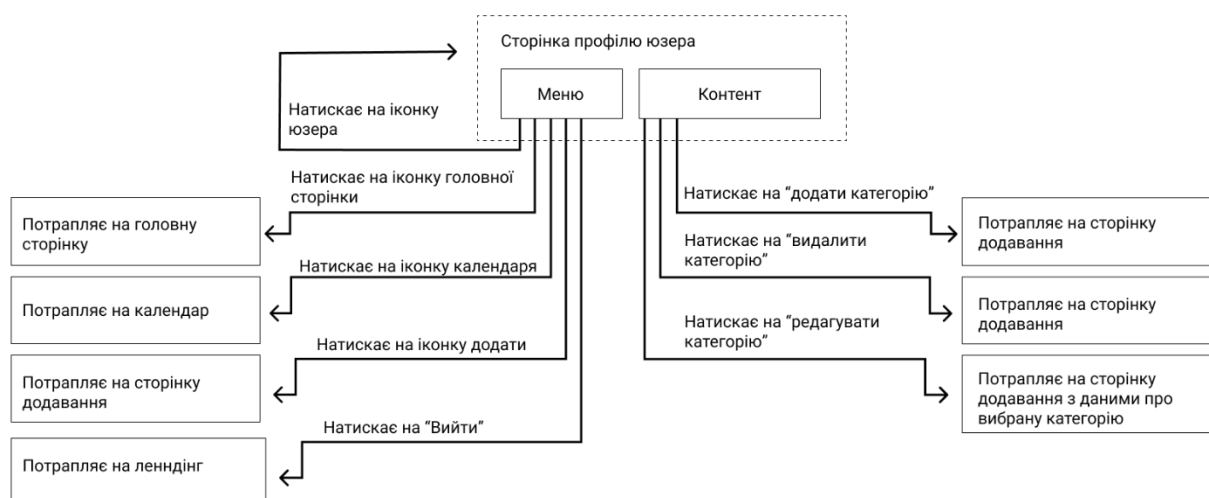


Рис. 15. Сценарій користувача – профіль юзера

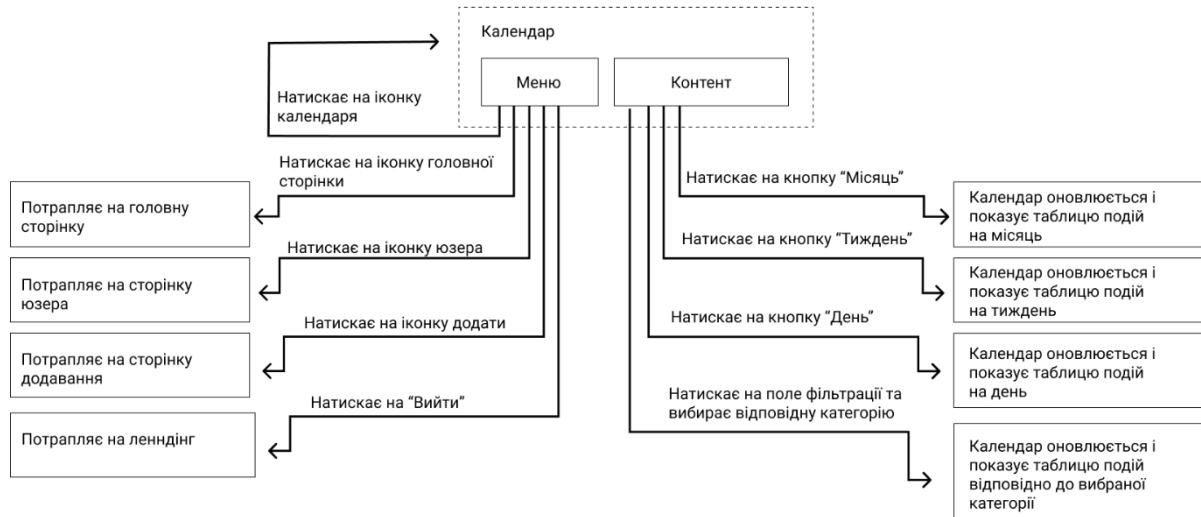


Рис. 16. Сценарій користувача – календар

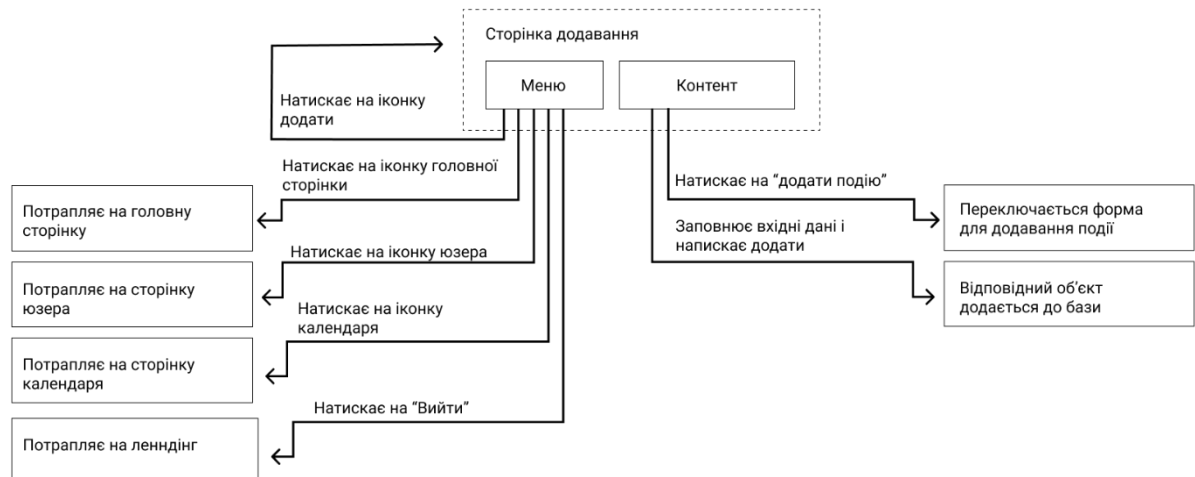


Рис. 17. Сценарій користувача – сторінка додавання

Отже, можемо побачити, що додаток візуально поділяється на 6 окремих компонентів-модулів:

1. Авторизація та аутентифікація
2. Меню користувача
3. Головна сторінка
4. Сторінка профілю юзера
5. Сторінка календаря

6. Сторінка додавання

3.2 Обґрунтування вибору засобів розробки

3.2.1 Front-end

Як будь-яке веб-застосування, даний сервіс має складатись із бекенду (для запитів до бази даних) та фронтенду (для взаємодії з користувачем).

Для реалізації фронтенду було використано стандартні засоби написання веб-сторінок: HTML, CSS, Javascript. Задля кращої структуризації проекту та поділення його на цілісні компоненти, було також вирішено використовувати бібліотеку React. До переваг цієї бібліотеки належать такі факти, як:

- Високий рівень гнучкості і максимальна чуйність.
- Віртуальна DOM, яка дозволяє впорядковувати документи форматів HTML в дерево, яке найкраще підходить веб-браузерам для аналізу різних елементів веб-додатку.
- Легко працює у поєднанні з ES6 або ES7 при високих навантаженнях.
- Чисто JavaScript-бібліотека з відкритим вихідним кодом, яка отримує безліч щоденних оновлень відповідно до відгуків розробників по всьому світу.
- Використання JSX синтаксису - розширення синтаксису JavaScript, яке зручно використовувати для опису інтерфейсу.

Для розробки клієнтської частини було використано середовище розробки WebStorm.

3.2.2 Back-end та бази даних

Для представлення бекенду і бази даних було використано Firebase – це одне з BaaS-рішень (Backend as Service), яке дає розробнику велику кількість можливостей. Це і сервер, і база даних, і хостинг, і аутентифікація в одній платформі, що є неймовірно зручно, адже по суті платформа надає програмісту API, який синхронізує дані додатки між клієнтами і зберігає їх в хмарному

сховищі. Даний підхід відрізняється від традиційної розробки додатків, яка, як правило, передбачає написання як інтерфейсної частини - клієнтської, так і серверної. З використанням Firebase фронтенд частина просто викликає кінцеві точки API, виставлені серверною базою, і бекенд частина автоматично виконує роботу.

Роботу даного підходу та відмінність він традиційного можна побачити нижче на Рисунку 18.

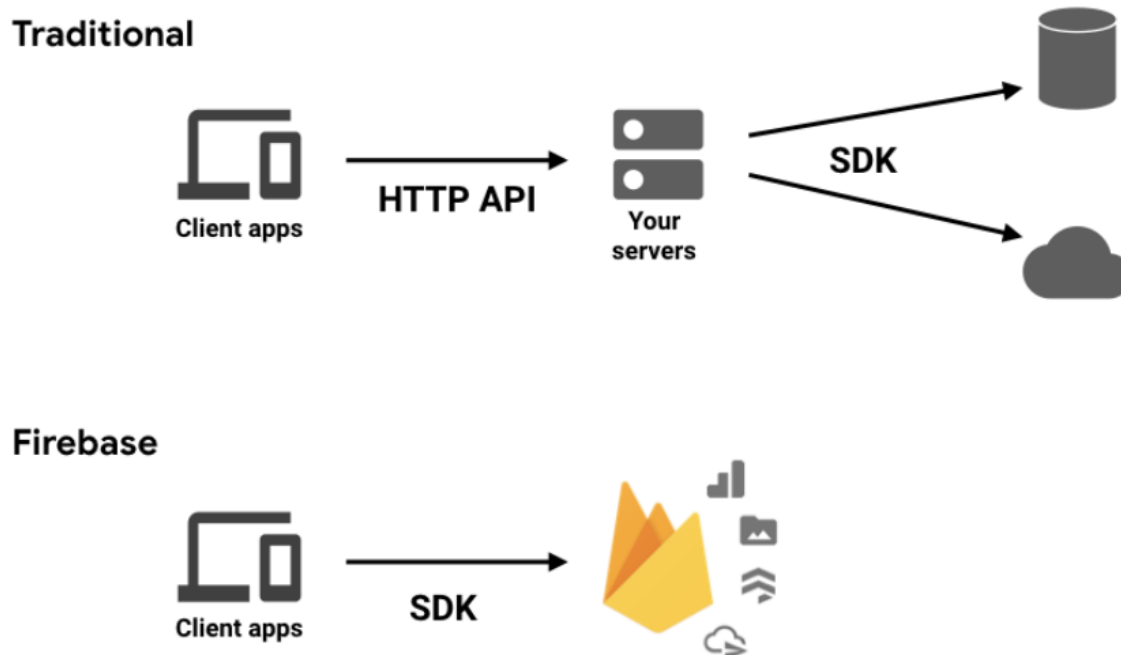


Рис. 18. Різниця структур сервісів з використанням Firebase та без [13]

Адміністративний доступ до функцій та сховищ забезпечує Firebase console.

До інших переваг цього сервісу можна включити також такі факти:

- Висока швидкість роботи програми.
- Надійна інфраструктура – відсутність збоїв в роботі.
- Зручна статистика, що дозволяє отримувати дані про дії користувачів і підтримувати зворотний зв'язок.
- Багатоплатформність – програма створена один раз і налаштована для роботи з усіма операційними системами.

- Проста масштабованість – зростання кількості користувачів не потребують змін у серверному коді.

База даних була реалізована через Firebase Database - це база даних NoSQL, яка має багато оптимізацій та функцій у порівнянні з більшістю реляційних баз даних. Вона включає гнучкі правила, що визначають структуру даних для забезпечення безпеки та гнучкості.

Об'єкти зберігаються у форматі JSON, що робить її використання простішим, ніж деякі бази даних SQL, для обробки даних як дерева. При додаванні нових даних до вашої бази даних, автоматично створюється вузол у існуючій структурі JSON із пов'язаним ключем.

3.3 Опис розробки програми

Розробка додатку здійснювалась за наступним переліком етапів:

1. Розподілення структури на конкретні компоненти, в результаті чого отримали 6 різних модулів:
 - a. Авторизація та аутентифікація,
 - b. Меню користувача
 - c. Головна сторінка
 - d. Сторінка профілю юзера
 - e. Сторінка календаря
 - f. Сторінка додавання
2. Для кожного модуля визначено, які вхідні дані потрібні, їх типи та обмеження:
 - a. Авторизація та аутентифікація:

Для входу у систему необхідно обов'язково ввести логін у представленні рядку електронної пошти та пароль також з типом рядку.

Для реєстрації необхідні такі дані: ім'я користувача – обов'язковий рядок, прізвище користувача - обов'язковий

рядок, можливий масив вибраних шаблонних категорій, електронна пошта користувача - обов'язковий рядок та пароль – обов'язковий рядок.

- b. Меню користувача: у даному блоці нема ніяких вхідних даних.
- c. Головна сторінка: у даному блоці нема ніяких вхідних даних.
- d. Сторінка профілю юзера: у даному блоці нема ніяких вхідних даних.
- e. Сторінка календаря: у даному блоці нема ніяких вхідних даних.
- f. Сторінка додавання:

Для форми «Додати категорію» необхідним полем є поле назви категорії – тип рядок. Також є додаткове поле опису категорії – тип рядок.

Для форми «Додати подію» необхідними полями є поле назви події – тип рядок, дата події – тип дата - та категорія події (вибір категорії серед категорій даного користувача) – тип рядок. Також є додаткове поле опису події – тип рядок та чекбокс на відмічення чи дана подія є терміновою – тип булеан.

3. Аналіз даних та розподіл на сутності.

Після збору всіх вхідних даних, їх було розподілено на сутності, у результаті чого вийшли 4 таблиці: таблиця користувача, таблиця категорій, таблиця події та таблиця цитати. На Рисунку 19 схематично представлено структуру кожної таблиці.

USER	CATEGORY	EVENT	QUOTE
# uid * email * firstName * lastName * categories [0..n]	# id * name * color * description * uid	# id * categoryId * dateTime * description * isDone * isHot * name * uid	# id * date * text * uid

Рис. 19. Схематичне представлення сутностей

4. Створення сховища в Firebase та налаштування середовища.
5. Налаштування авторизації у Firebase.
6. Створення відповідних таблиць з даними у сховищі Firebase.

Нижче на Рисунках 20, 21, 22 та 23 представлена реалізація сутностей у нереляційній базі даних Firebase як колекцій.

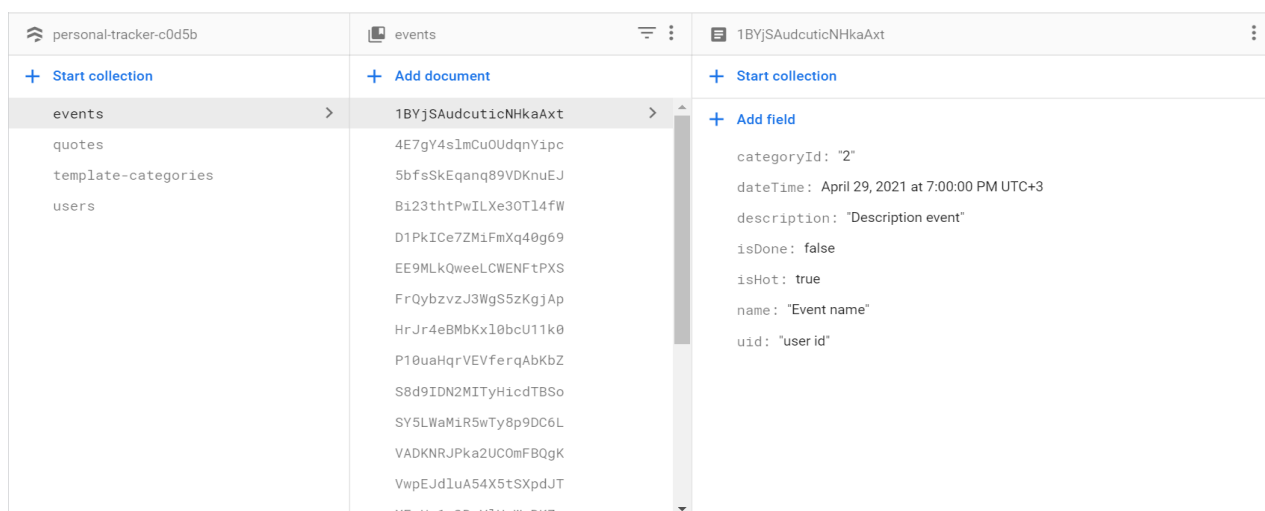


Рис. 20. Колекція подій

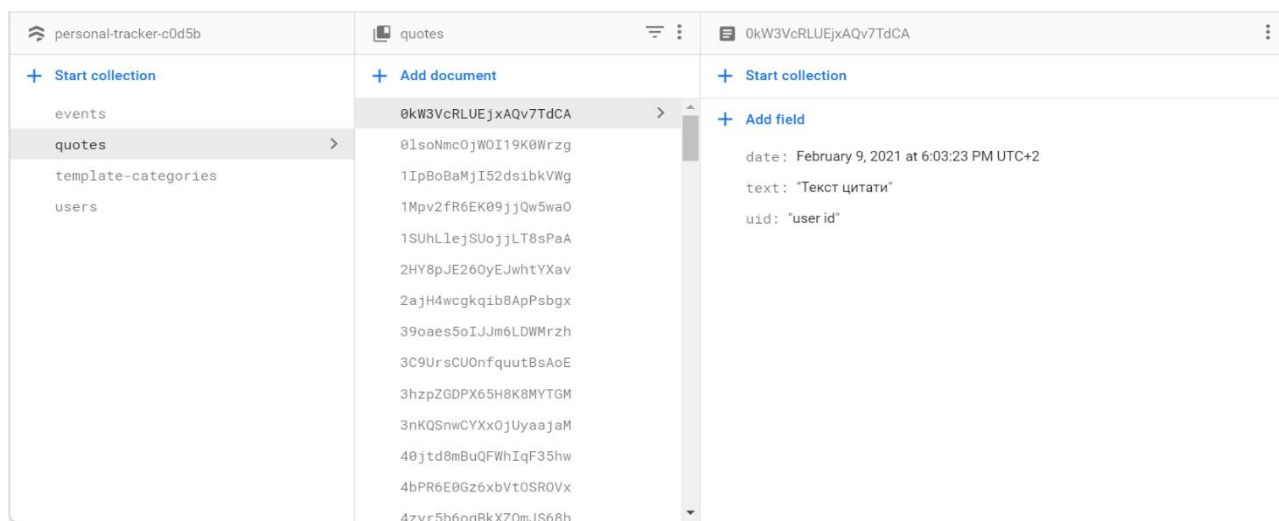


Рис. 21. Колекція цитат

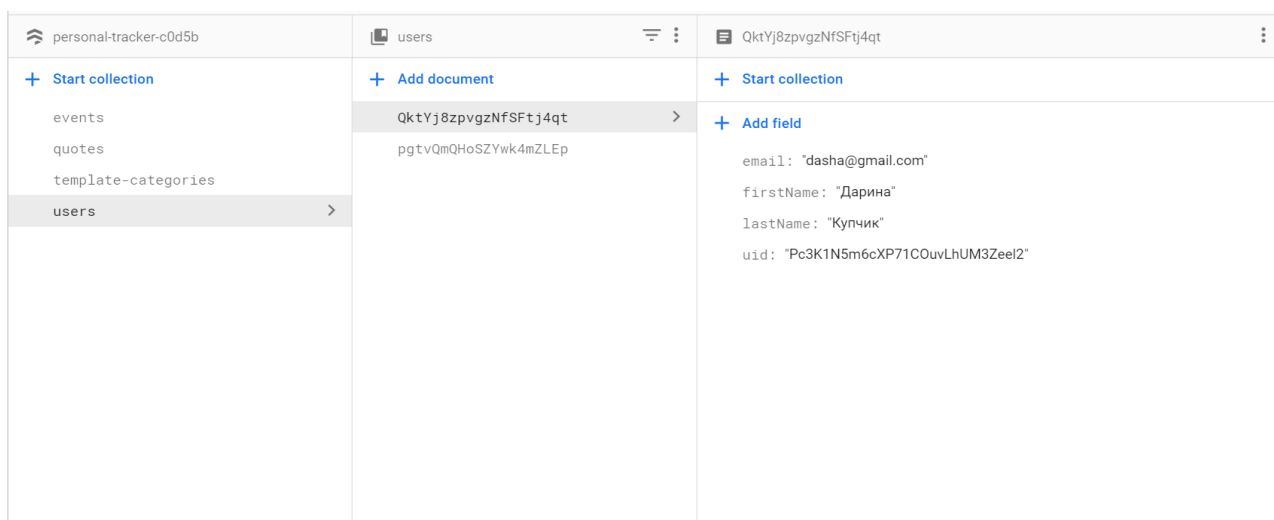


Рис 22. Колекція цитат

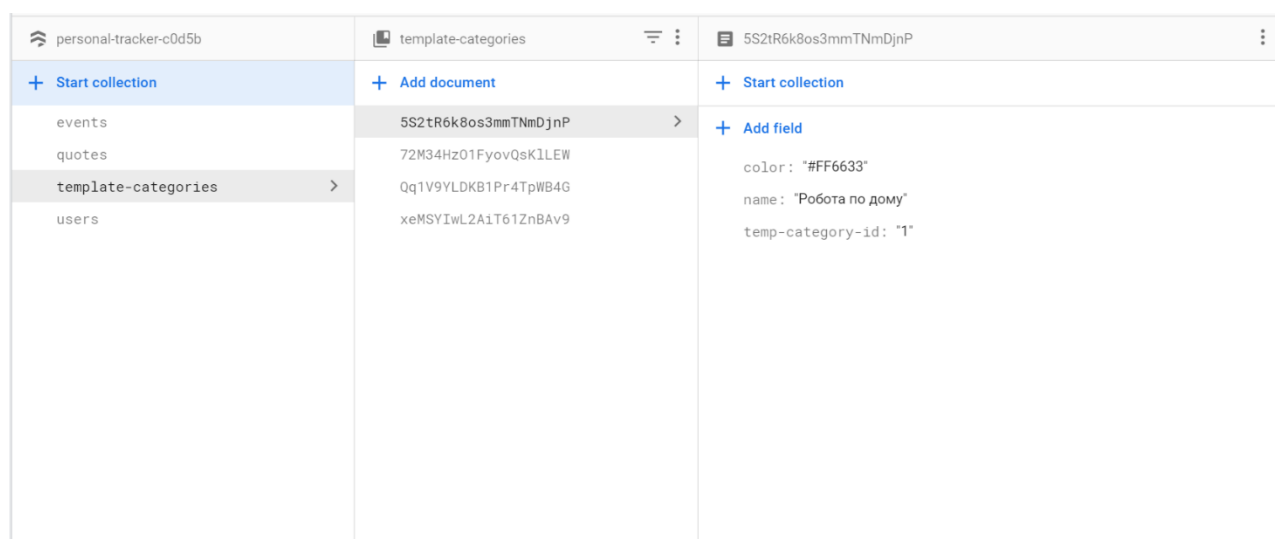


Рис 23. Колекція категорій

7. Верстання всіх компонентів в тому ж порядку, в якому вони наведені в пункті 1.
8. Підключення Firebase до клієнтської частини.

На даному етапі було створена файл **firebase.js**, у якому було підключення до створеного у пункті 4 сховища.

firebase.js:

```
import firebase from 'firebase'
import 'firebase/auth'

const firebaseConfig = {
  apiKey: "-",
  authDomain: "personal-tracker-c0d5b.firebaseio.com",
  projectId: "personal-tracker-c0d5b",
  storageBucket: "personal-tracker-c0d5b.appspot.com",
  messagingSenderId: "-",
```

```

    appId: "- ",
    measurementId: "-"
  });

const fire = firebase.initializeApp(firebaseConfig);
firebase.analytics();

export const auth = fire.auth();
export default fire;

```

9. Створення та налаштування Redux сховища.

Задля ефективного використання даних користувача, покращення продуктивності застосунку та уникнення повторів запитів на сервер було використано Redux сховище.

Під час розробки було створено окрему папку **store** у якій зберігаються всі actions та reducers сховища. Налаштування самого сховища виглядає так:

```

const composeEnhancers =
  typeof window === 'object' &&
  window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__
    ? window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__({})
    : compose

const store = createStore(
  rootReducer,
  {},
  composeEnhancers(applyMiddleware(thunkMiddleware))
)

```

10. Реалізація головної сторінки та ініціалізації Redux сховища.

Всі необхідні запити на сервер було реалізовано при першому запуску застосування у файлі **App.js**.

У даному файлі робляться такі запити:

- запит на отримання всіх завдань користувача: метод для отримання всіх завдань користувача було реалізовано як власний хук **useTasks**, який приймає на вхід id користувача та на виході ми отримуємо всі події даного юзера **useTasks**:

```

export const useTasks = (userId) => {
  const [tasks, setTasks] = useState([]);
  let unsubscribe = fire
    .firestore()
    .collection('events')
    .where('uid', '==', userId)

  useEffect(() => {
    unsubscribe = unsubscribe.onSnapshot(snapshot => {
      const newTasks = snapshot.docs.map(task => ({

```

```

        id: task.id,
        ...task.data()
      )))

      setTasks(newTasks)
    })

  }, [userId])

  return {tasks};
}

```

- запит на отримання всіх цитат користувача: метод для отримання всіх цитат користувача було реалізована як власний хук **useUserQuotes**, який приймає на вхід id користувача та на виході ми отримуємо всі цитати даного юзера.

useUserQuotes:

```

export const useUserQuotes = (userId) => {
  const [userQuotes, setUserQuotes] = useState([]);
  let unsubscribe = fire
    .firestore()
    .collection('quotes')
    .where('uid', '==', userId)

  useEffect(() => {
    unsubscribe = unsubscribe.onSnapshot(snapshot => {
      const newUserQuotes = snapshot.docs.map(quote => ({
        id: quote.id,
        ...quote.data()
      }))

      setUserQuotes(newUserQuotes)
    })

  }, [userId])

  return {userQuotes};
}

```

- запит на отримання шаблонних категорій: метод для отримання всіх шаблонних категорій було реалізована як власний хук **useTemplateCategories**, який нічого не приймає на вхід та на виході ми отримуємо всі шаблоні категорії.

useTemplateCategories:

```

export const useTemplateCategories = () => {
  const [templateCategories, setTemplateCategories] = useState([]);

  useEffect(() => {

```

```

    let unsubscribe = fire
      .firestore()
      .collection('template-categories')

    unsubscribe.onSnapshot(snapshot => {
      const newCategories = snapshot.docs.map(category => ({
        id: category.id,
        ...category.data()
      })
    })

    setTemplateCategories(newCategories)
  })
}, [])

return {templateCategories};
}

```

- запит на отримання інформації про користувача: метод для отримання всієї інформації про користувача було реалізовано як власний хук **useUserInfo**, який приймає на вхід id користувача та на виході ми отримуємо потрібну інформацію.

useUserInfo:

```

export const useUserInfo = (userId) => {
  const [userInfo, setUserInfo] = useState({});
  let unsubscribe = fire
    .firestore()
    .collection('users')
    .where('uid', '==', userId)

  useEffect(() => {
    unsubscribe = unsubscribe.onSnapshot(snapshot => {
      const newUserInfo = snapshot.docs.map(user => ({
        id: user.id,
        ...user.data()
      })
    })

    setUserInfo(newUserInfo[0])
  })

  }, [userId])

  return {userInfo};
}

```

Всі дані, які прийшли з сервера, зберігаються в створеному нами сховищі у пункті 10 задля легкого перевикористання на будь-якій іншій сторінці.

11. Реалізація функціоналу авторизації та аутентифікації.

Задля реалізації входу в систему, було створено метод, який кидає запит на серверну частину, валідує вхідні данні і віддає відповідь. Даний метод виглядає так:

```
auth
  .signInWithEmailAndPassword(emailValue, password)
  .then(res => {
    dispatch(setUser(res.user.uid))
    dispatch(setEmail(emailValue))

    window.localStorage.setItem('uid', res.user.uid);
  })
  .catch(error => {
    switch (error.code) {
      case 'auth/email-already-in-use':
      case 'auth/invalid-email':
        setEmailError(error.message);
        break;

      case 'auth/weak-password':
        setPasswordError(error.message);
        break;

      default:
        setPasswordError('Oops.. something went wrong..try again
:)' );
        break;
    }
  })
```

Для реалізації реєстрації було розроблено аналогічний метод:

```
auth.createUserWithEmailAndPassword(emailValue, password)
  .then(res => {
    dispatch(setUser(res.user.uid))
    dispatch(setEmail(emailValue))

    window.localStorage.setItem('uid', res.user.uid);
  })
  .catch(error => {
    switch (error.code) {
      case 'auth/invalid-emailValue':
      case 'auth/user-disabled':
      case 'auth/user-not-found':
        setEmailError(error.message);
        break;

      case 'auth/wrong-password':
        setPasswordError(error.message);
        break;

      default:
        setPasswordError('Oops.. something went wrong..try again
:)' );
        break;
    }
  })
```

12. Реалізація функціоналу головної сторінки користувача.

Для головної сторінки потрібно отримати нагадування користувача, передбачення та справи на сьогодні.

Задля отримання події, яка буде нагадуванням для користувача, було використано фільтрацію за умовою, що день події є найближчим днем, але не є сьогоднішнім днем. В результаті отримуємо такий метод:

```
const getReminder = () => {
  const filteredTasks = tasks.filter(task => task.dateTime.toDate() >
new Date()
    && task.dateTime.toDate().getDay() !== (new Date()).getDay())
  filteredTasks.sort(function (a, b) {
    return a.dateTime.toDate() >= b.dateTime.toDate()
  })

  return filteredTasks.length > 0
    ? setReminder(`${filteredTasks[0].name}:
${filteredTasks[0].description}`)
    : setReminder('Нічого термінового нема');
}
```

Задля знаходження всіх сьогоднішніх подій було використано фільтрацію за умовою, що день події є сьогоднішнім днем. Результат отримуємо таким способом:

```
setTodayTasks(tasks.filter(task => task.dateTime.toDate().toDateString()
=== (new Date()).toDateString()));
```

Відсоток виконаних на сьогоднішній день завдань вираховується за допомогою такого методу:

```
const changePercentage = () => {
  const doneTasksLength = todayTasks.filter(task => task.isDone).length
  setPercentage(todayTasks.length === 0 || doneTasksLength === 0
    ? 0
    : (doneTasksLength * 100 / todayTasks.length).toFixed(1));
}
```

13. Реалізація функціоналу сторінки профіля користувача.

Для сторінки профілю потрібно отримати всі категорії даного користувача та статистику про його події.

Задля отримання категорій було використано звичайну фільтрацію по тій інформації користувача, яка у нас вже зберігається у сховищі.

Задля отримання статистики користувача було також відфільтровано вже існуючі дані. Візуальна частина графіку подій користувача була реалізована з використанням бібліотеки Nivo [14].

14. Реалізація функціоналу сторінки додавання.

Для сторінки додавання категорії нічого не потрібно завчасно отримувати з бази даних.

Для сторінки додавання події потрібно отримати всі категорії даного користувача, щоб реалізувати селекту у формі тільки зі значень цих категорій.

Додавання категорії відбувається за допомогою методу **fireAddCategory**, який виглядає наступним чином:

```
export const fireAddCategory = (id, uid, email, firstName, lastName,
categories, color, nextCategoryId, categoryName, categoryDesc) =>
  fire.firestore().collection("users").doc(id)
    .update({
      email: email,
      firstName: firstName,
      lastName: lastName,
      categories: [...categories, {color: color, id:
nextCategoryId.toString(), name: categoryName, desc: categoryDesc}],
      uid: uid
    })
    .then(() => {
      alert("Категорія була успішно додана!");
    })
    .catch((error) => {
      alert("Error: " + error);
    });
```

Додавання події відбувається за допомогою методу **fireAddEvent**, який виглядає наступним чином:

```
export const fireAddEvent = (eventName, eventDesc, eventCategory, eventDate,
eventIsHot, uid) =>
  fire.firestore().collection("events").add({
    name: eventName,
    description: eventDesc,
    categoryId: eventCategory.toString(),
    isHot: eventIsHot,
    dateTime: firebase.firestore.Timestamp.fromDate(new
Date(eventDate)),
    uid: uid
  })
  .then(() => {
    alert("Подія була успішно опублікована!");
  })
  .catch((error) => {
    alert("Error: " + error);
  });
```

15. Реалізація функціоналу сторінки календаря.

Для сторінки календаря зі сховища потрібно отримати всі категорії даного користувача та всі його події.

Фільтрація подій відбувалась за допомогою методу **changeEventCategory**, який виглядає наступним чином:

```
const changeEventCategory = selectedOption => {
  setEventCategory(selectedOption)

  switch (selectedOption.value) {
    case 'All':
      setEvents(allEvents)
      break;

    case 'Hot':
      setEvents(allEvents.filter(task => task.isHot))
      break;

    default:
      setEvents(allEvents.filter(task => task.categoryId ===
selectedOption.value))
  }
}
```

Візуальна частина календаря була реалізована з використанням бібліотеки Fullcalendar [15].

3.4 Створення об'єктів і розробка головної програми

У програмі можливе створення користувача при авторизації або аутентифікації, створення категорії та створення певної події.

При створенні користувача сервер присилає нам його токен, який надалі зберігається в localStorage та є ідентифікатором користувача.

При створенні нової категорії або події клієнтська частина відправляє всі необхідні дані на створення нового документу, документ автоматично створюється та додається до бази.

3.5 Опис файлів даних та інтерфейсу програми

Результуючий інтерфейс базується на схемах-інтерфейсу, які були розроблені у пункті 3.1.2, адже вони слугували прототипом розробки.

Коли користувач відкриває веб-застосунок, він опиняється на головній сторінці, яка зображена на Рисунку 24.

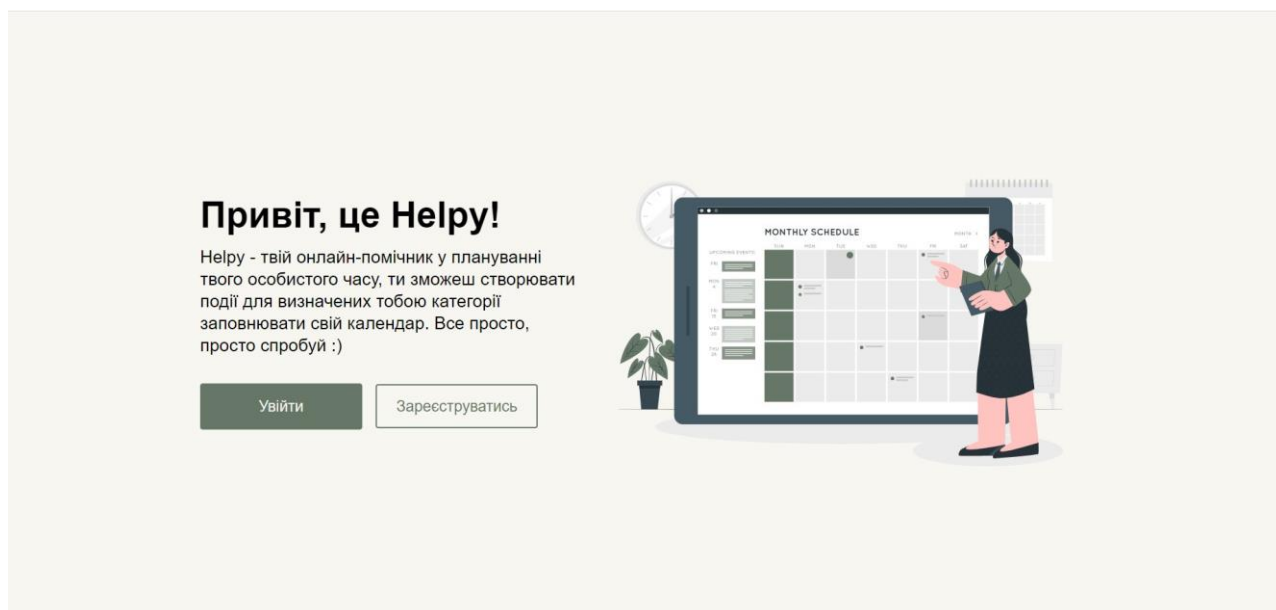
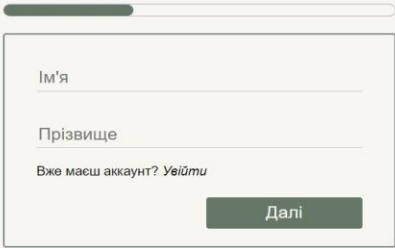


Рис. 24. Головна сторінка застосунку

Після цього у користувача є можливість увійти або зареєструватись натиснувши на відповідні кнопки. Натиснувши на кнопку «Увійти» користувач побачить форму, яка зображена на Рисунку 25, а при виборі реєстрації користувачу потрібно буде заповнити три послідовні форми як на рисунках 26, 27 та 28.



Рис. 25 Сторінка входу



A registration form titled "Ім'я" (Name) and "Прізвище" (Surname). It includes a link "Вже маєш аккаунт? Увійти" (Already have an account? Log in) and a "Далі" (Next) button. A progress bar at the top shows the first step is completed.

Ім'я

Прізвище

Вже маєш аккаунт? [Увійти](#)

Далі

Рис. 26. Сторінка реєстрації – крок 1



A registration form titled "Можливо ти хочеш додати певні категорії для тренінгу?" (Perhaps you want to add certain categories for training?). It includes checkboxes for "Робота по дому" (Home work), "Навчання" (Education), and "Дозвілля" (Hobbies). It also includes a link "Вже маєш аккаунт? Увійти" (Already have an account? Log in) and buttons "Назад" (Back) and "Далі" (Next). A progress bar at the top shows the second step is completed.

Можливо ти хочеш додати певні категорії для тренінгу?

☒ Робота по дому

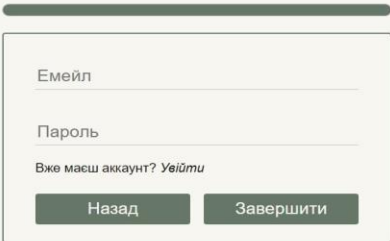
☒ Навчання

☒ Дозвілля

Вже маєш аккаунт? [Увійти](#)

Назад Далі

Рис. 27. Сторінка реєстрації – крок 2



A registration form titled "Емейл" (Email) and "Пароль" (Password). It includes a link "Вже маєш аккаунт? Увійти" (Already have an account? Log in) and buttons "Назад" (Back) and "Завершити" (Finish). A progress bar at the top shows the third step is completed.

Емейл

Пароль

Вже маєш аккаунт? [Увійти](#)

Назад Завершити

Рис. 28. Сторінка реєстрації – крок 3

Після того як користувач зареєструється, він побачить головну сторінку, яка зображена на Рисунку 29.



Рис. 29. Головна сторінка авторизованого користувача без даних

Якщо користувач вже користується сервісом, тобто має певні дані, та увійде у систему, то також побачить головну сторінку, але вже заповнену потрібними даними, як на Рисунку 30:

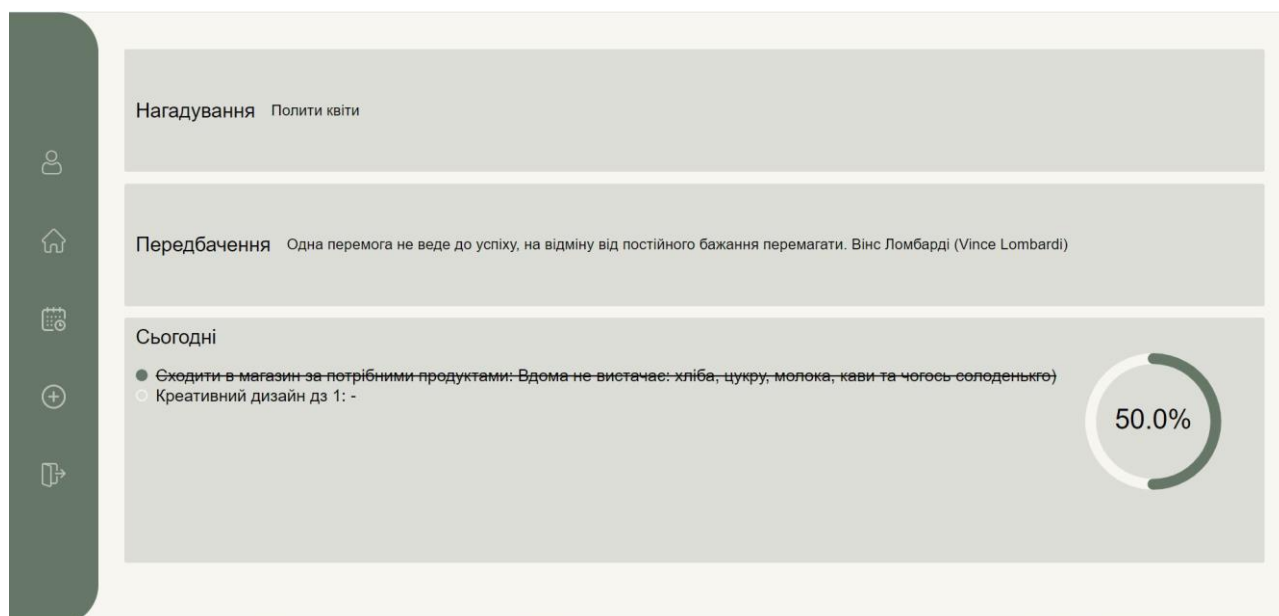


Рис. 30. Головна сторінка авторизованого користувача з даними

Можемо побачити, що у кожного користувача зліва є навігаційне меню.

Якщо користувач натисне на іконку «Користувач», то отримає сторінку свого профілю (Рис. 31), на якому зображена інформація про нього, його категорії та статистика подій.

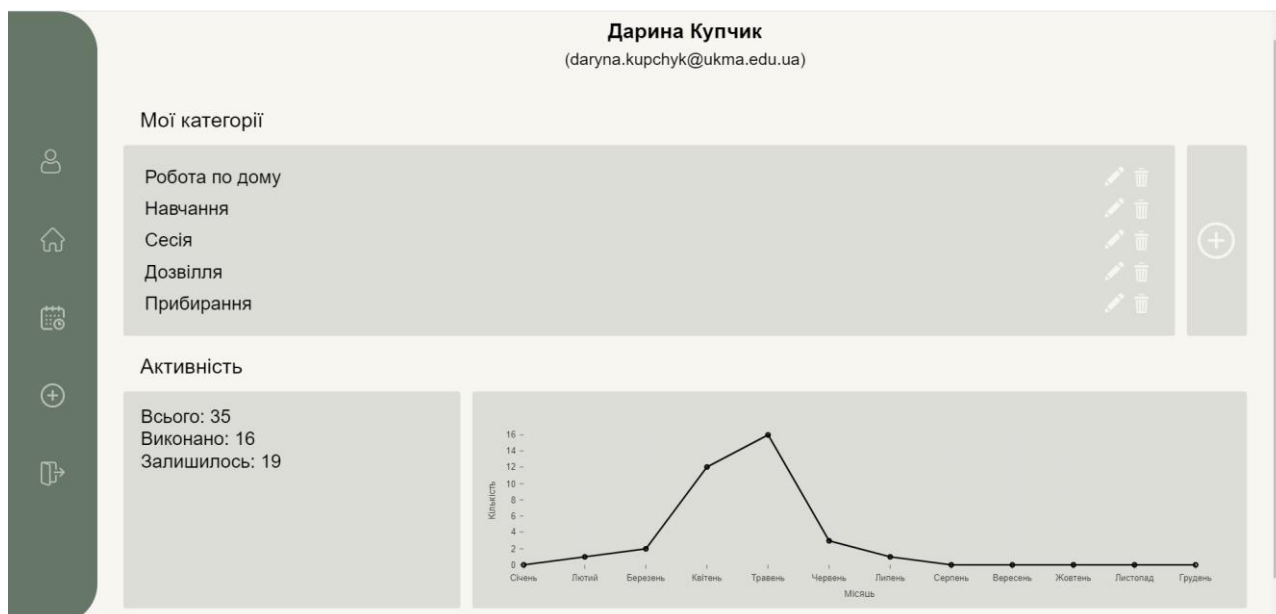


Рис. 31. Сторінка профіля користувача

Якщо користувач натисне на іконку «Календар», то побачить сторінку календаря, який заповнений всіма подіями користувача, як показано на Рисунку 32:

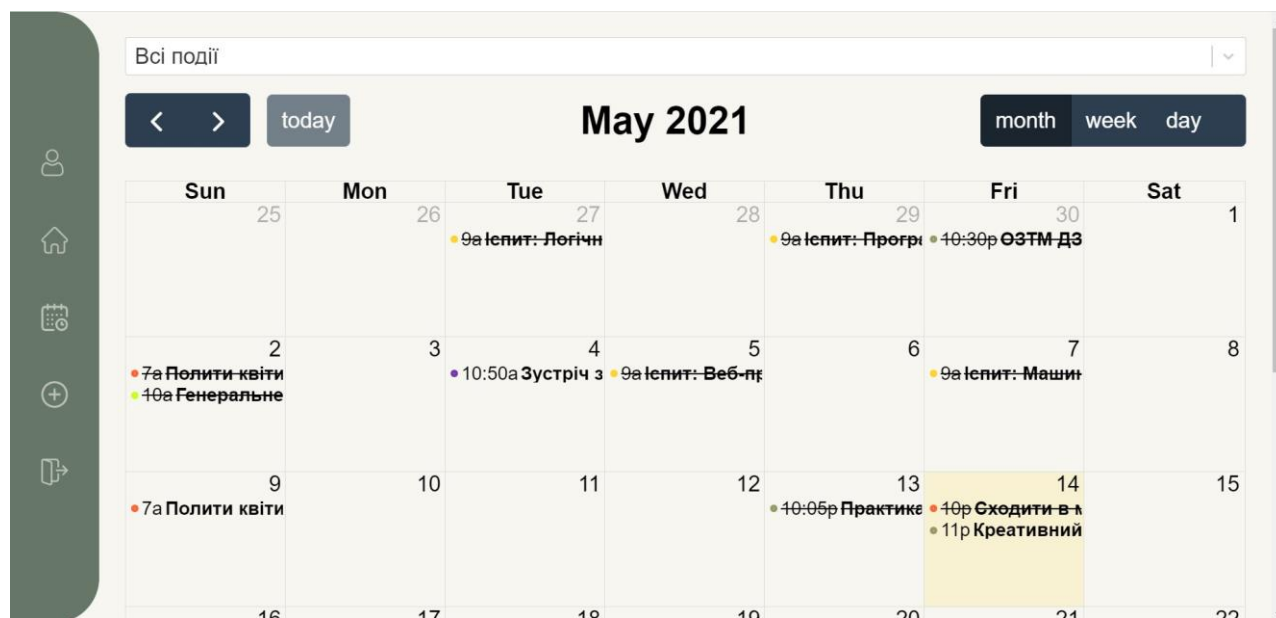


Рис. 32. Сторінка календаря користувача – дефолтна фільтрація

Якщо користувач хоче побачити події у календарі тільки певної категорії, то він може вибрати відповідну категорію зверху і події в календарі оновляться автоматично, як показано на Рисунку 33:

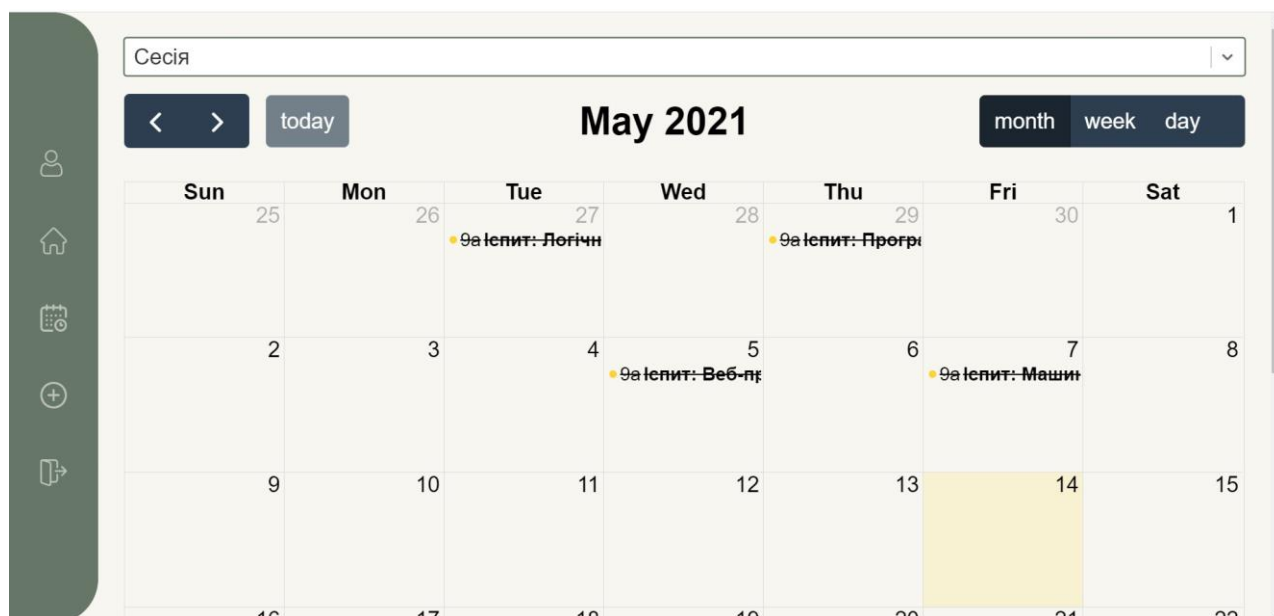


Рис. 33. Сторінка календаря користувача –фільтрація по категорії «Сесія»

Якщо користувач натисне на іконку «Додати», то перейде на сторінку додавання категорії або події. Форми додавання категорії та події зображені на Рисунку 34 та Рисунку 35 відповідно.

Рис. 34. Сторінка додавання – форма «Додати категорію»

Додати категорію

Додати подію

Назва

ДД.ММ.ГГГГ --:--

Категорія..

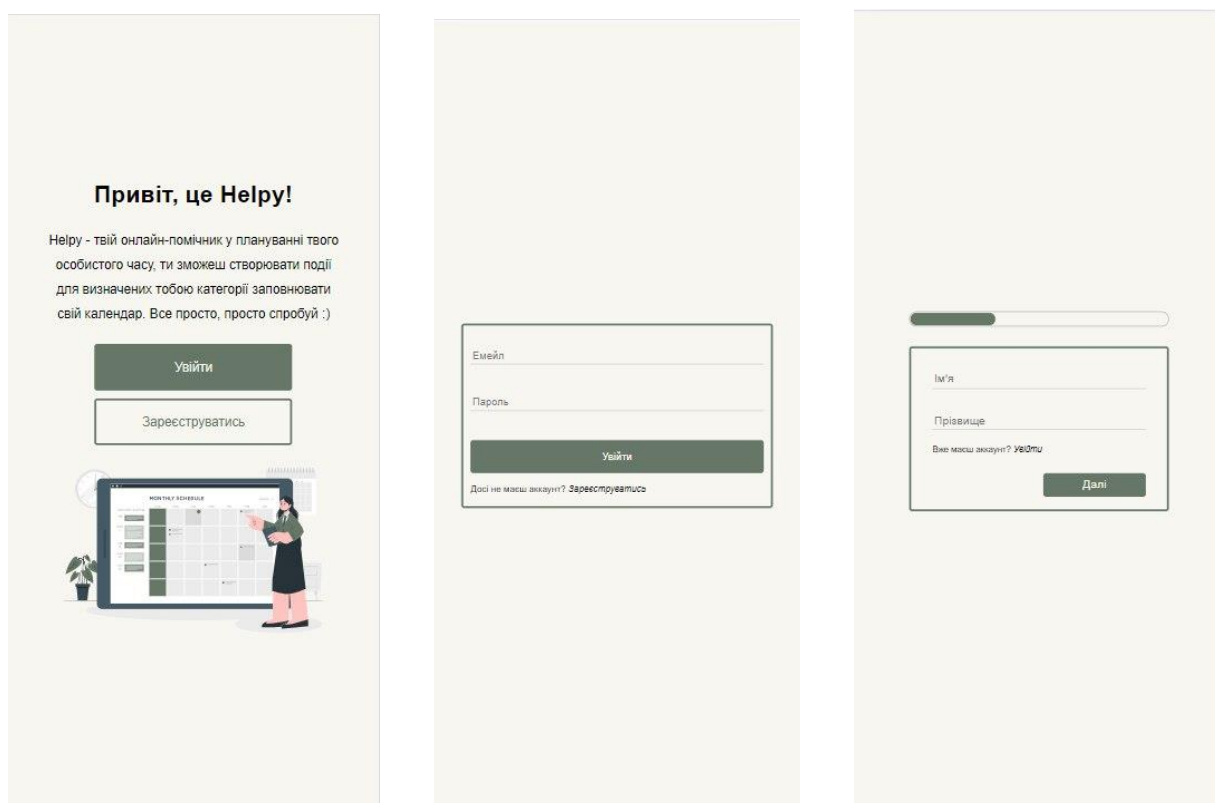
Опис

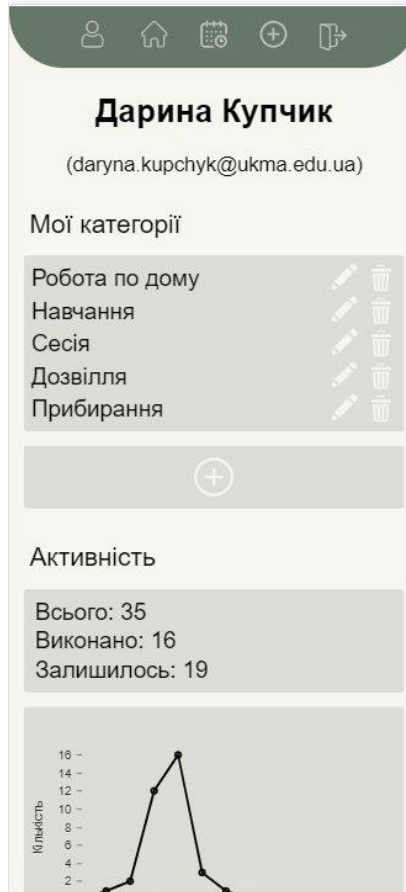
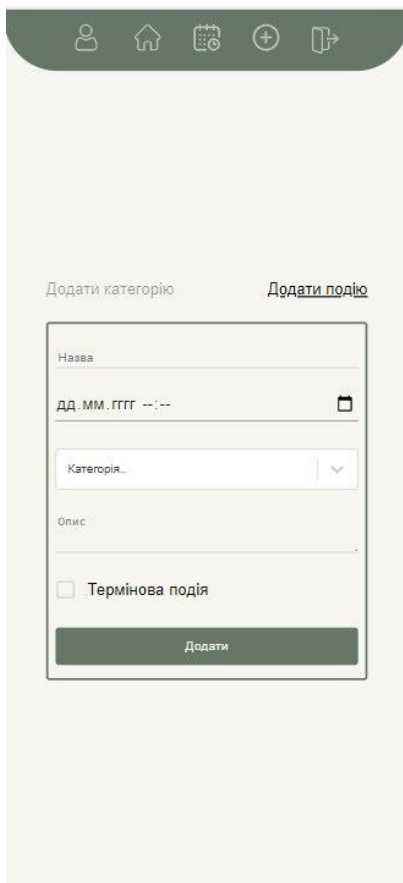
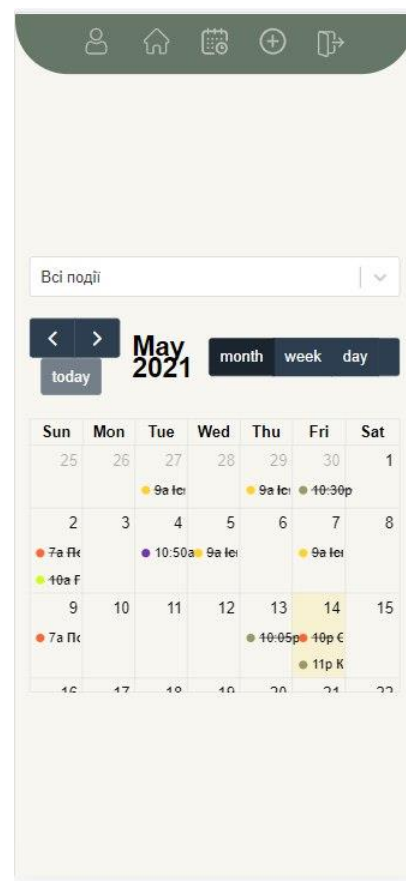
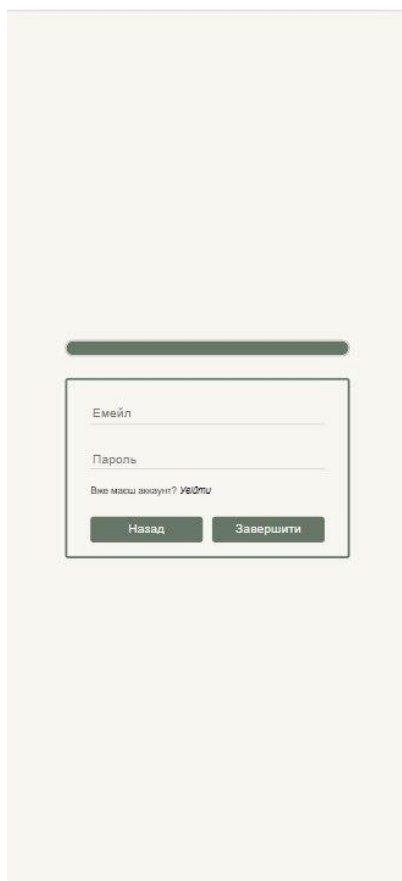
☐ Термінова подія

Додати

Рис. 35. Сторінка додавання – форма «Додати подію»

Нижче наведено приклади інтерфейсу у мобільній версії:





3.6 Тестування програми і результати її виконання

При тестуванні програми було виконано мануальне тестування, під час виконання якого було виписано всі можливі дії користувача у Таблицю 4 та перевірено, що кожна функція виконується без будь-яких перешкод.

№	Функціонал	Етапи досягнення результату	Результат тестування
1	Вхід у систему з головної сторінки	<ol style="list-style-type: none"> 1. Користувач заходить на головну сторінку 2. Натискає на кнопку «Вхід» і потрапляє на сторінку з формочкою входу 3. Вводить правильні електронну пошту та пароль 4. Входить в систему 	Виконано успішно
2	Вхід у систему зі сторінки реєстрації	<ol style="list-style-type: none"> 1. Користувач заходить на головну сторінку 2. Натискає на кнопку «Реєстрація» і потрапляє на сторінку з першим кроком для реєстрації 3. Натискає «Увійти» 4. Потрапляє на форму входу і виконує всі кроки з пункту №1 	Виконано успішно
3	Реєстрація в систему з головної сторінки	<ol style="list-style-type: none"> 1. Користувач заходить на головну сторінку 2. Натискає на кнопку «Реєстрація» і потрапляє на сторінку з першим кроком для реєстрації 3. Вводить правильні ім'я та прізвище 4. Натискає «Далі» 5. Вибирає потрібні категорії 6. Натискає «Далі» 	Виконано успішно

		<p>7. Вводить правильні електронну пошту та пароль</p> <p>8. Натискає «Завершити»</p> <p>9. Входить в систему</p>	
4	Реєстрація в систему зі сторінки входу	<p>1. Користувач заходить на головну сторінку</p> <p>2. Натискає на кнопку «Вхід» і потрапляє на сторінку входу</p> <p>3. Натискає «Зареєструватись»</p> <p>4. Потрапляє на перший крок реєстрації і виконує всі кроки з пункту №3</p>	Виконано успішно
5	Відмічення будь-якої події з сьогоднішніх як виконана	<p>1. Користувач потрапляє на головну сторінку</p> <p>2. Натискає на будь-яку подію, які є в переліку блоку «Сьогодні»</p> <p>3. Подія, на яку було натиснуто, позначається як завершена(перекреслюється) та відсоток виконаних справ відповідно змінюється</p>	Виконано успішно
5	Видалення категорії користувача, якщо не існує подій цієї категорії	<p>1. Користувач потрапляє на сторінку профілю користувача</p> <p>2. Натискає на іконку видалення категорії, у якої нема подій</p> <p>3. З'являється алерт з питанням чи справді користувач хоче видалити категорію.</p> <p>4. Натискає «Так»</p> <p>5. З'являється оповіщення про успішне видалення</p> <p>6. Категорія видаляється</p>	Виконано успішно

6	Оповіщення про неможливе видалення категорії користувача, якщо існує хоча б одна подій цієї категорії	<ol style="list-style-type: none"> 1. Користувач потрапляє на сторінку профілю користувача 2. Натискає на іконку видалення категорії, у якої є подій 3. З'являється оповіщення про неможливе видалення 4. Категорія не видаляється 	Виконано успішно
7	Редагування категорії	<ol style="list-style-type: none"> 1. Користувач потрапляє на сторінку профілю користувача 2. Натискає на іконку редагування вибраної категорії і потрапляє на сторінку редагування 3. Змінює потрібні дані 4. Натискає «Редагувати» 5. Категорія успішно редагується 	Виконано успішно
8	Додавання категорії зі сторінки профіля	<ol style="list-style-type: none"> 1. Користувач потрапляє на сторінку профілю користувача 2. Натискає на іконку додавання категорії і потрапляє на сторінку Додавання категорії 3. Виконує всі кроки з пункту №8 	Виконано успішно
9	Додавання категорії зі сторінки додавання	<ol style="list-style-type: none"> 1. Користувач потрапляє на сторінку додавання категорії або події 2. Вводить всі необхідні дані про категорію 3. Натискає «Додати» 4. Категорія додається 	Виконано успішно

10	Додавання події	<ol style="list-style-type: none"> 1. Користувач потрапляє на сторінку додавання категорії або події 2. Натискає на «Додати подію» 3. Вводить всі необхідні дані про подію 4. Натискає «Додати» 5. Подія додається 	Виконано успішно
11	Фільтрація по вибраній категорії у календарі	<ol style="list-style-type: none"> 1. Користувач потрапляє на сторінку календаря 2. Вибирає зверху необхідну категорію для фільтрації 3. Події у календарі оновлюються відповідно до вибраної категорії 	Виконано успішно
12	Вихід із системи	<ol style="list-style-type: none"> 1. Користувач натискає на іконку виходу у меню 2. Користувач автоматично виходить із системи та потрапляє на головну сторінку неавторизованого юзера 	Виконано успішно

Таблиця. 4. Результати тестування

Висновки

У результаті виконання цієї роботи поставлена задача була виконана: було створено застосунок, який дозволяє користувачу ефективно відстежувати свій час та справи за допомогою таких функцій як створення та збереження своїх категорій подій, створення та збереження будь-яких справ та відстеження їх у календарі. Таким чином було досягнуто поставлену мету щодо розробки працездатного сайту із простим та зручним інтерфейсом і корисними для будь-якої людини можливостями.

Розроблена система хоч і виконує всі функціональні вимоги та має напрямки для подальшого вдосконалення:

1. Додавання фільтрації по категоріям у загальній статистиці користувача задля наглядності прогресу саме по категоріям.
2. Під'єднання сторонніх ресурсів, які користуються попитом, наприклад, Google Calendar, задля синхронізації всіх можливих справ користувача в розроблений додаток.
3. Додання подій або категорій за допомогою голосу, що прискорить користувачам процес створення нових справ та занесення їх у застосунок.

Список використаної літератури

1. Стаття зі статистикою 2021 року щодо тайм-менеджменту [Електронний ресурс] <https://development-academy.co.uk/news-tips/time-management-statistics-2021-research/>
2. Стаття про причини знехтування використання трекерів часу [Електронний ресурс] <https://www.productive.io/blog/psychological-barrier-time-tracking/>
3. Стаття про 5 переваг використання трекеру часу та справ [Електронний ресурс] <https://work.chron.com/five-good-effects-time-management-workplace-27491.html>
4. Стаття про позитивні впливи використання трекерів часу та справ на ментальне здоров'я [Електронний ресурс] <https://paintedbrain.org/mental-health/how-is-time-management-important-for-your-mental-health/>
5. Стаття про статистику трекерів часу та справ [Електронний ресурс] <https://collegeinfo geek.com/productivity-apps/>
6. Посилання на головну сторінку застосунку Todoist [Електронний ресурс] <https://todoist.com/>

7. Приклад інтерфейсу застосунку Todoist [Електронний ресурс]
https://get.todoist.help/hc/article_attachments/360005842740/text-formatting_web.png
8. Посилання на головну сторінку застосунку TickTick [Електронний ресурс]
<https://ticktick.com/>
9. Приклад інтерфейсу застосунку TickTick [Електронний ресурс]
<https://startpack.ru/application/ticktick>
10. Посилання на головну сторінку застосунку Notion [Електронний ресурс]
<https://www.notion.so/>
11. Приклад інтерфейсу застосунку Notion [Електронний ресурс]
https://s0.rbk.ru/v6_top_pics/resized/945xH/media/img/6/60/755906727933606.png
12. Посилання на головну сторінку застосунку Figma [Електронний ресурс]
<https://www.figma.com/>
13. Стаття про загальну інформацію про Firebase [Електронний ресурс]
<https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0>
14. Посилання на головну сторінку бібліотеки Nivo [Електронний ресурс]
<https://nivo.rocks/>
15. Посилання на головну сторінку бібліотеки Fullcalendar [Електронний ресурс]
<https://fullcalendar.io/>

Додатки

App.js – головний файл для ініціалізації:

```
import React, {useEffect, useState} from "react";
import {BrowserRouter as Router} from 'react-router-dom'
import {useDispatch, useSelector} from "react-redux";

import './App.css';

import AppRouter from "./AppRouter";
import {useTasks, useTemplateCategories, useUserInfo, useUserQuotes} from
"./shared/hooks";
import {
  setCategories,
  setEmail,
  setFirstName,
  setLastName,
  setNextCategoryId, setQuote,
  setTasks, setUsedCategoryColors,
  setUser
} from "./store/actions/actions";
import {fireAddQuote} from "./shared/firebaseCalls";
import {quotes} from "./shared/constants/utilsConstants";

function App() {
  const dispatch = useDispatch()

  const user = useSelector(state => state.mainReducer.uid) ||
window.localStorage.getItem('uid');

  const {tasks} = useTasks(user)
  const {templateCategories} = useTemplateCategories()
  const {userInfo} = useUserInfo(user)
  const {userQuotes} = useUserQuotes(user)

  useEffect(() => {
    if (user && userInfo && Object.keys(userInfo).length !== 0) initStore()
  }, [user, tasks, userInfo, templateCategories, userQuotes]);

  const initStore = () => {
    dispatch(setUser(userInfo.uid))

    dispatch(setFirstName(userInfo.firstName))
    dispatch(setEmail(userInfo.email))
    dispatch(setLastName(userInfo.lastName))
    dispatch(setCategories(userInfo.categories))
    dispatch(setTasks(tasks))

    dispatch(setNextCategoryId(Math.max(...userInfo.categories.map(category
=> category.id)) + 1))
    dispatch(setUsedCategoryColors(userInfo.categories.map(category =>
category.color)))

    if(userQuotes) getQuote()
  }

  const getQuote = () => {
    let quote;
    const todayQuotes = userQuotes.filter(quote =>
quote.date.toDate().toDateString() === (new Date()).toDateString())
```

```

    if(todayQuotes.length === 0) {
      quote = quotes[Math.floor(Math.random() * quotes.length)]
      fireAddQuote(user, quote);
    } else {
      quote = todayQuotes[0]
    }

    dispatch(setQuote(quote))
  }

  return (
    <Router>
      <div className="App">
        <AppRouter/>
      </div>
    </Router>
  );
}

export default App;

```

firebase.js – підключення Firebase:

```

import firebase from 'firebase'
import 'firebase/auth'

const firebaseConfig = {
  apiKey: "AIzaSyDUkWoVdRJmHLbcohFITr8fL-m7Q4imBeA",
  authDomain: "personal-tracker-c0d5b.firebaseio.com",
  projectId: "personal-tracker-c0d5b",
  storageBucket: "personal-tracker-c0d5b.appspot.com",
  messagingSenderId: "77856629054",
  appId: "1:77856629054:web:f1fd21c8769b0caf018c6f",
  measurementId: "G-7RCH1TF6WC"
};

const fire = firebase.initializeApp(firebaseConfig);
firebase.analytics();

export const auth = fire.auth();
export default fire;

```

AppRouter.js – налаштування шляхів веб-застосунку:

```

import React from "react";
import {Switch, Route, Redirect} from 'react-router-dom'
import {useSelector} from "react-redux";

import './App.css';

import {HOME_PAGE_ROUTE, LANDING_PAGE_ROUTE} from
"./shared/constants/routesConstants";
import {privateRoutes, publicRoutes} from "./shared/routes";

function AppRouter() {
  const user = useSelector(state => state.mainReducer.uid) ||
window.localStorage.getItem('uid');

  return user
    ? (<Switch>
      {privateRoutes.map(({path, Component}) => <Route path={path} exact

```

```

component={Component}/>)}
      <Redirect to={HOME_PAGE_ROUTE}/>
    </Switch>
    : (<Switch>
      {publicRoutes.map(({path, Component}) => <Route path={path} exact
component={Component}/>)}
      <Redirect to={LANDING_PAGE_ROUTE}/>
    </Switch>)
  }

export default AppRouter;

```

LandingPage.jsx – головна сторінка, на яку потрапляє неавторизований користувач:

```

import React from "react";
import {Link} from "react-router-dom";

import styles from './LandingPage.module.css'

function LandingPage() {
  return (
    <div className={styles['container']}>

      <div className={styles['text-column']}>
        <h2>Привіт, це Helpy!</h2>
        <p>Helpy - твій онлайн-помічник у плануванні твого особистого
часу, ти зможеш створювати події для визначених тобою категорії заповнювати свій
календар. Все просто, просто спробуй :)</p>

        <div className={styles['buttons']}>
          <Link classname={styles['button-link']} to={`/auth/sign-
in`}>
            <button className={styles['sign-in']}>Увійти</button>
          </Link>

          <Link classname={styles['button-link']} to={`/auth/sign-
up/step-1`}>
            <button>Зареєструватись</button>
          </Link>

        </div>

      </div>

    </div>
  );
}

export default LandingPage;

```

MainPage.jsx – головна сторінка, на яку потрапляє авторизований користувач:

```

import React, {useEffect, useState} from "react";
import {useSelector} from "react-redux";

```

```

import {
  CircularProgressbar,
  buildStyles
} from "react-circular-progressbar";
import "react-circular-progressbar/dist/styles.css";

import styles from './MainPage.module.css'
import Menu from "../Menu/Menu";
import {mainPageSelector} from "../mainPageSelector";
import {useUserQuotes} from "../../shared/hooks";
import {fireChangeEventIsDone} from "../../shared/firebaseCalls";

function MainPage() {
  const {userQuotes} = useUserQuotes(window.localStorage.getItem('uid'))
  const {uid, templateCategories, tasks, quote} =
    useSelector(mainPageSelector)

  const [reminder, setReminder] = useState('Нічого термінового нема')
  const [todayTasks, setTodayTasks] = useState([])
  const [percentage, setPercentage] = useState(0);

  useEffect(() => {
    setTodayTasks(tasks.filter(task => task.dateTime.toDate().toDateString()
    === (new Date()).toDateString())));

    changePercentage();
    getReminder();
  }, [tasks, templateCategories, userQuotes]);

  const changeTaskIsDone = (i) => {
    todayTasks[i].isDone = !todayTasks[i].isDone;
    setTodayTasks([...todayTasks]);
    changePercentage();

    fireChangeEventIsDone(
      todayTasks[i].id,
      todayTasks[i].categoryId,
      todayTasks[i].dateTime,
      todayTasks[i].description,
      todayTasks[i].isDone,
      todayTasks[i].isHot,
      todayTasks[i].name,
      uid)
  }

  const changePercentage = () => {
    const doneTasksLength = todayTasks.filter(task => task.isDone).length
    setPercentage(todayTasks.length === 0 || doneTasksLength === 0
      ? 0
      : (doneTasksLength * 100 / todayTasks.length).toFixed(1));
  }

  const getReminder = () => {
    const filteredTasks = tasks.filter(task => task.dateTime.toDate() > new
    Date()
      && task.dateTime.toDate().getDay() !== (new Date()).getDay())
    filteredTasks.sort(function (a, b) {
      return a.dateTime.toDate() >= b.dateTime.toDate()
    })

    return filteredTasks.length > 0
      ?
    setReminder(`${filteredTasks[0].name}${filteredTasks[0].description ? ': ' +

```

```

filteredTasks[0].description : '' `)`
      : setReminder('Нічого термінового нема');
    }

    return (
      <div className={styles['container']}>

        <Menu/>

        <div className={styles['content']}>

          <div className={styles['reminder']}>
            <h2>Нагадування</h2>
            <div>{reminder}</div>
          </div>

          <div className={styles['quote']}>
            <h2>Передбачення</h2>
            <div>{quote}</div>
          </div>

          <div className={styles['today']}>
            <h2>Сьогодні</h2>

            <div>
              {todayTasks.length !== 0
                ? <div className={styles['tasks']}>
                  {todayTasks.map((task, index) =>
                    <div key={index} className={styles['task']}
                      onClick={() =>
changeTaskIsDone(index)}>
![circle]({`./images/${task.isDone)

```

```

    );
}

export default MainPage;

```

UserInfo.jsx – сторінка профілю користувача:

```

import React, {useEffect, useState} from "react";
import {useSelector} from "react-redux";
import {Link, useHistory} from "react-router-dom";
import {ResponsiveLine} from "@nivo/line";

import styles from './UserInfo.module.css'

import Menu from "../Menu/Menu";
import {selector} from "../userInfoSelector";
import {taskService} from "../../shared/taskService";
import {fireUpdateCategory} from "../../shared/firebaseCalls";
import {useUserInfo} from "../../shared/hooks";

function UserInfo() {
  let history = useHistory()

  const {userInfo} = useUserInfo(window.localStorage.getItem('uid'))
  const {email, firstName, lastName, categories, tasks} =
    useSelector(selector)

  const [name, setName] = useState('-')
  const [taskData, setTaskData] = useState({})

  useEffect(() => {
    setName(`${firstName} ${lastName}`)
    setTaskData(taskService(tasks))
  }, [firstName, lastName]);

  const toAddPage = () => history.push('/add')

  const deleteCategory = (id, name) => {
    if (tasks.filter(task => task.categoryId === id).length > 0) {
      alert('Ви не можете видалити дану категорію, адже до неї вже
прив'язані якісь події.')
      return;
    }

    if (window.confirm(`Ви впевнені що хочете видалити категорію
"${name}"?`)) {
      const newCategories = [...categories]
      const index = categories.findIndex(x => x.id === id);
      newCategories.splice(index, 1);

      fireUpdateCategory(userInfo.id, userInfo.uid, email, firstName,
lastName, newCategories, true)
    }
  }

  return (
    <div className={styles['container']}>

      <Menu/>

      <div className={styles['content']}>
        <h2>{name}</h2>
        <p>({email})</p>

```

```

<div className={styles['wrapper']}>
  <p className={styles['title']}>Мої катеропії</p>

  <div className={styles['my-categories-container']}>
    <div className={styles['my-categories']}>

      {categories.map(category =>
        <div className={styles['category']}>
          <p>{category.name}</p>
          <div className={styles['icons']}>

            <Link to={{
              pathname: '/update',
              state: {
                propCategoryId: category.id,
                propCategoryColor:
category.color,

                propCategoryName: category.name,
                propCategoryDesc: category.desc,
              }
            }}>
              

            </Link>

            
deleteCategory(category.id, category.name)} alt="delete"/>
            </div>
          </div>
        )}
      </div>

    <div onClick={toAddPage} className={styles['add-button-
wrapper']}>
      
    </div>

  </div>
</div>

<div className={styles['wrapper']}>
  <p className={styles['title']}>Активність</p>

  <div className={styles['statistics-container']}>
    <div className={styles['statistics']}>
      <div>Всього: <span>{taskData.total}</span></div>
      <div>Виконано: <span>{taskData.done}</span></div>
      <div>Залишилось: <span>{taskData.left}</span></div>
    </div>

    <div className={styles['chart']}>
      <ResponsiveLine
        data={taskData.data}
        colors={['#000']}
        margin={{top: 40, right: 50, bottom: 40, left:
50}}

        xScale={{type: 'point'}}
        yScale={{type: 'linear', min: 'auto', max:
'auto', stacked: true, reverse: false}}
        yFormat=" >-.2f"
        axisTop={null}

```



```

axisRight={null}
axisBottom={{
  orient: 'bottom',
  tickSize: 5,
  tickPadding: 5,
  tickRotation: 0,
  legend: 'Місяць',
  legendOffset: 36,
  legendPosition: 'middle'
}}
axisLeft={{
  orient: 'left',
  tickSize: 5,
  tickPadding: 5,
  tickRotation: 0,
  legend: 'Кількість',
  legendOffset: -40,
  legendPosition: 'middle'
}}
pointSize={5}
pointColor={{theme: 'background'}}
pointBorderWidth={2}
pointBorderColor={{from: 'serieColor'}}
pointLabelYOffset={-12}
areaBlendMode="screen"
areaBaselineValue={40}
areaOpacity={0}
useMesh={true}
/>
</div>
</div>
</div>
</div>
</div>
);
}

export default UserInfo;

```

Menu.jsx – меню:

```

import React from "react";
import {useHistory, Link} from "react-router-dom";
import {useDispatch} from "react-redux";

import styles from './Menu.module.css'

import {
  setCategories,
  setEmail,
  setFirstName,
  setLastName,
  setNextCategoryId,
  setTasks, setUsedCategoryColors,
  setUser
} from "../../store/actions/actions";

function Menu() {
  const dispatch = useDispatch()
  const history = useHistory();

```

```

const logout = () => {
  window.localStorage.removeItem('uid');

  dispatch(setUser(null))
  dispatch(setFirstName(''))
  dispatch(setEmail(''))
  dispatch(setLastName(''))
  dispatch(setCategories([]))
  dispatch(setTasks([]))
  dispatch(setNextCategoryId(0))
  dispatch(setUsedCategoryColors([]))

  history.push('/main');
}

return (
  <div className={styles['menu']}>

    <div className={styles['icons']}>
      <Link to={`\user`}>
        
      </Link>

      <Link to={`\home`}>
        
      </Link>

      <Link to={`\calendar`}>
        
      </Link>

      <Link to={`\add`}>
        
      </Link>

    </div>

  </div>
);
}

export default Menu;

```

Add.jsx – сторінка додавання категорії або події:

```

import React, {useEffect, useState} from "react";
import Select from 'react-select'
import {useDispatch, useSelector} from "react-redux";

import Checkbox from "@material-ui/core/Checkbox";
import FormControlLabel from "@material-ui/core/FormControlLabel";
import TextField from "@material-ui/core/TextField";
import {withStyles} from "@material-ui/core";

import styles from './Add.module.css'

import Menu from "../Menu/Menu";
import {addSelector} from "../userInfoSelector";

```

```

import {useUserInfo} from "../../shared/hooks";
import {setNextCategoryId, setUsedCategoryColors} from
"../../store/actions/actions";
import {colors} from "../../shared/constants/utilsConstants";
import {fireAddCategory, fireAddEvent} from "../../shared/firebaseCalls";

const GreenCheckbox = withStyles({
  root: {
    color: 'rgba(103, 119, 103, 0.2)',
    '&$checked': {
      color: 'rgba(103, 119, 103, 1)',
    },
  },
  checked: {},
})(props) => <Checkbox color="default" {...props} />;

function Add() {
  const dispatch = useDispatch()

  const {userInfo} = useUserInfo(window.localStorage.getItem('uid'))
  const {email, firstName, lastName, categories, nextCategoryId,
usedCategoryColors} = useSelector(addSelector)

  const [isCategoryFormSelected, setIsCategoryFormSelected] = useState(true)

  const [categoryName, setCategoryName] = useState('')
  const [categoryDesc, setCategoryDesc] = useState('')
  const [categoryNameError, setCategoryNameError] = useState('')

  const [eventName, setEventName] = useState('')
  const [eventDesc, setEventDesc] = useState('')
  const [eventCategory, setEventCategory] = useState({value: 'Категорія..',
label: 'Категорія..'})
  const [eventCategories, setEventCategories] = useState([])
  const [eventDate, setEventDate] = useState('')
  const [eventIsHot, setEventIsHot] = useState(false)

  const [eventNameError, setEventNameError] = useState('')
  const [eventCategoryError, setEventCategoryError] = useState('')
  const [eventDateError, setEventDateError] = useState('')

  useEffect(() => {
    setEventCategories(categories.map(item => ({value: item.id, label:
item.name})))
  }, [categories]);

  const changeCategoryName = e => setCategoryName(e.target.value)
  const changeCategoryDesc = e => setCategoryDesc(e.target.value)

  const changeEventName = e => setEventName(e.target.value)
  const changeEventCategory = selectedOption =>
setEventCategory(selectedOption)
  const changeEventDesc = e => setEventDesc(e.target.value)
  const changeEventDate = e => setEventDate(e.target.value)
  const changeEventIsHot = _ => setEventIsHot(!eventIsHot)
  const changeSelected = () =>
setIsCategoryFormSelected(!isCategoryFormSelected)

  const addCategory = (e) => {
    e.preventDefault()

    if (categoryName === '') {
      setCategoryNameError('Назва категорії повинна обов\`язково бути');
      return;

```

```

    }

    const color = generateColor();

    fireAddCategory(
      userInfo.id,
      userInfo.uid,
      email,
      firstName,
      lastName,
      categories,
      color,
      nextCategoryId,
      categoryName,
      categoryDesc)

    dispatch(setNextCategoryId(nextCategoryId + 1))
    dispatch(setUsedCategoryColors([...usedCategoryColors, color]))

    resetCategoryForm();
  }

  const generateColor = () => {
    let color = colors[Math.floor(Math.random() * colors.length)];

    while (usedCategoryColors.includes(color))
      color = colors[Math.floor(Math.random() * colors.length)];

    return color;
  }

  const addEvent = (e) => {
    e.preventDefault()

    resetAllEventErrors()

    if (eventName === '') {
      setEventNameError('Назва події повинна обов\'язково бути');
      return;
    }

    if (eventDate === '') {
      setEventDateError('Дата та час події повинна обов\'язково бути')
      return;
    }

    if (eventCategory.value === 'Категорія..') {
      setEventCategoryError('Виберіть одну зі своїх категорій')
      return;
    }

    fireAddEvent(eventName, eventDesc, eventCategory.value, eventDate,
    eventIsHot, window.localStorage.getItem('uid'));

    resetEventForm();
  }

  const resetAllEventErrors = () => {
    setEventNameError('');
    setEventCategoryError('');
    setEventDateError('');
  }

  const resetEventForm = () => {

```

```

    setEventName('');
    setEventCategory('');
    setEventDesc('');
    setEventDate('');

    document.getElementById('addEventForm').reset();
  }

  const resetCategoryForm = () => {
    setCategoryName('');
    setCategoryDesc('');

    document.getElementById('addCategoryForm').reset();
  }

  return (
    <div className={` ${styles['container']} ${styles['add-container']}`}>

      <Menu/>

      <div className={styles['content']}>

        <div>
          <p onClick={changeSelected}
            className={isCategoryFormSelected ? styles['selected'] :
styles['not-selected']}>
            Додати категорію</p>
          <p onClick={changeSelected}
            className={!isCategoryFormSelected ? styles['selected'] :
styles['not-selected']}>Додати
            подію</p>
        </div>

        {isCategoryFormSelected
          ? <form id='addCategoryForm'>
            <input type="text"
              placeholder="Назва"
              onChange={changeCategoryName}/>
            <p className={styles['error']}>{categoryNameError}</p>

            <textarea placeholder="Опис"
              onChange={changeCategoryDesc}/>

            <button type="submit"
onClick={addCategory}>Додати</button>
          </form>
          : <form id='addEventForm'>
            <input type="text"
              placeholder="Назва"
              onChange={changeEventName}/>
            <p className={styles['error']}>{eventNameError}</p>

            <div className={styles['dateTime']}>
              <TextField
                id="datetime-local"
                type="datetime-local"
                defaultValue={eventDate}
                className={styles['textField']}
                onChange={changeEventDate}
                InputLabelProps={{
                  shrink: true,
                }}
              />
            </div>

```

```

        <p className={styles['error']}>{eventDateError}</p>

        <div className={styles['select']}>
            <Select value={eventCategory}
                    onChange={changeEventCategory}
                    options={eventCategories}
            />
        </div>
        <p className={styles['error']}>{eventCategoryError}</p>

        <textarea placeholder="Опис"
            onChange={changeEventDesc}/>

        <FormControlLabel style={{fontSize: '2.5em'}}
            control={
                <GreenCheckbox
                    checked={eventIsHot}
                    onChange={changeEventIsHot}
                    name="checkedG"
                />
            }
            label="Термінова подія"
        />

        <button type="submit" onClick={addEvent}>Додати</button>
    </form>
  }

</div>

</div>
);
}

export default Add;

```

Calendar.jsx – сторінка календаря:

```

import React, {useEffect, useState} from "react";
import {useDispatch, useSelector} from "react-redux";
import {useHistory} from "react-router";
import Select from "react-select";

import FullCalendar from "@fullcalendar/react";
import dayGridPlugin from "@fullcalendar/daygrid";

import styles from './Calendar.module.css'

import Menu from "../Menu/Menu";
import {calendarSelector} from "../calendarSelector";
import {setChosenEventId} from "../../store/actions/actions";

export default function Calendar() {
  const history = useHistory();
  const dispatch = useDispatch();

  const {categories, tasks} = useSelector(calendarSelector)

  const [allEvents, setAllEvents] = useState([]);
  const [events, setEvents] = useState([]);
  const [eventCategory, setEventCategory] = useState({value: 'Всі події',
label: 'Всі події'})
  const [eventCategories, setEventCategories] = useState([])

```

```

useEffect(() => {
  const eventsFromTasks = tasks.map(task => ({
    id: task.id,
    title: `${task.isHot ? '🔥' : ''} ${task.name}`,
    date: task.dateTime.toDate(),
    categoryId: task.categoryId,
    isHot: task.isHot,
    color: categories.find(el => el.id === task.categoryId).color,
    className: `${task.isDone ? styles['done'] : ''}`,
  }))
  )

  setEvents(eventsFromTasks)
  setAllEvents(eventsFromTasks)
  setEventCategories([
    {value: 'All', label: 'Bci podii'},
    {value: 'Hot', label: '🔥 Hot'},
    ...categories.map(item => ({value: item.id, label: item.name}))])
}, [tasks, categories]);

const changeEventCategory = selectedOption => {
  setEventCategory(selectedOption)

  switch (selectedOption.value) {
    case 'All':
      setEvents(allEvents)
      break;

    case 'Hot':
      setEvents(allEvents.filter(task => task.isHot))
      break;

    default:
      setEvents(allEvents.filter(task => task.categoryId ===
selectedOption.value))
  }
}

const onEventClick = (info) => {
  info.jsEvent.preventDefault();
  dispatch(setChosenEventId(info.event._def.publicId))
  history.push('/detailed')
}

return (
  <div className={styles['container']}>

    <Menu/>

    <div className={styles['content']}>

      <div className={styles['select']}>
        <Select value={eventCategory}
          onChange={changeEventCategory}
          options={eventCategories}
          styles={{
            menu: provided => ({...provided, zIndex: 9999}),
            outlineColor: 'red'
          }}
          theme={(theme) => ({
            ...theme,

```

```

        borderRadius: 3,
        colors: {
            ...theme.colors,
            primary25: 'rgba(103, 119, 103, 0.25)',
            primary: '#677767',
        },
    })),
    />
</div>

<FullCalendar
  defaultView="listWeek"
  events={events}
  eventClick={onEventClick}
  headerToolbar={{
    start: 'prev,next today',
    center: 'title',
    end: 'dayGridMonth,dayGridWeek,dayGridDay,dayGridlist'
  }}
  plugins={[dayGridPlugin]}
  />
</div>
</div>
);
}

```