МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВЕТЕРИНАРНОЇ МЕДИЦИНИ ТА БІОТЕХНОЛОГІЙ ІМ. С.З. ҐЖИЦЬКОГО
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

першого (бакалаврського) рівня вищої освіти

на тему: **«Розробка веб-додатку для моніторингу особистих щоденних задач»**

Виконав: студент 4 курсу групи Кн-41

Спеціальності 122 «Комп'ютерні науки»
(шифр і назва)

Щадило Тарас Ярославович
(Прізвище та ініціали)

Керівник: к.т.н., доцент Падюка Роман Іванович
(Прізвище та ініціали)

ДУБЛЯНИ-2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА
ІНФОРМАЦІЙНИХ ТЕХНЕОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Перший (бакалаврський) рівень вищої освіти
Спеціальність 122 «Комп'ютерні науки»

«ЗАТВЕРДЖУЮ»

Завідувач кафедри_____

д.т.н., проф. А. М. Тригуба
«____» _____2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу студенту

Щадило Тарасу Ярославовичу

1. Тема роботи: «Розробка веб-додатку для моніторингу особистих щоденних задач»

Керівник роботи Падюка Роман Іванович, доцент
затверджені наказом по університету від 25.02.2025 року № 123/к-с.

2. Строк подання студентом роботи 10.06.2025 р.
3. Вихідні дані до роботи: *вимоги до проектування інформаційних систем; методика проектування інформаційних систем; технічне завдання на проектування веб-застосунку для моніторингу особистих щоденних задач .*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)
Вступ.
1. Аналіз предметної області.
2. Постановка задачі та планування проекту
3. Реалізація веб-застосунку для моніторингу особистих щоденних задач
4. Охорона праці та безпека в надзвичайних ситуаціях
Висновки та пропозиції.
Список використаної літератури.
5. Перелік ілюстраційного матеріалу (з точним зазначенням обов'язкових креслень): *актуальність теми, мета та завдання роботи, аналіз існуючих рішень, технологічна база проекту, функціонал застосунку, архітектурні рішення, реалізація інтерфейсу, алгоритми та сценарії, результати розробки.*

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3	*Падюка Р.І., доцент кафедри ІТ*		
4	*Городецький І.М., доцент кафедри інженерної механіки*		

7. Дата видачі завдання 26 лютого 2025 р.

Календарний план

№ з/п	Назва етапів дипломного проекту	Терміни виконання етапів роботи	Примітка
1	*Написання першого розділу*	*26.02.25-20.03.25*	
2	*Виконання другого розділу та аркушів ілюстраційного матеріалу до нього*	*21.03-15.04.25*	
3.	*Виконання третього розділу та аркушів ілюстраційного матеріалу до нього*	*16.04-30.04.25*	
4.	*Написання розділу «Охорона праці»*	*01-15.05.25*	
5.	*Завершення оформлення розрахунково-пояснювальної записки та аркушів ілюстраційного матеріалу*	*15-30.05.25*	
6.	*Завершення роботи вцілому*	*01 - 10.06.25*	

Студент _____ Щадило Т.Я.
(підпис)

Керівник роботи _____Падюка Р.І.
(підпис)

УДК 004.9 : 631.1

Розробка веб-додатку для моніторингу особистих щоденних задач.

Щадило Т.Я. Кафедра ІТ – Дубляни, Львівський НУВМБ ім. С.З. Гжицького, 2025.

Кваліфікаційна робота: 67 с. текст. част., 32 рис., 2 табл., 11 арк. ілюстраційного матеріалу, 17 джерел.

Розроблено сучасний веб-застосунок для управління персональними завданнями, підкреслюється її актуальність у сучасному оцифрованому середовищі, де ефективне управління часом має вирішальне значення.

Проаналізовано існуючі рішення, такі як Todoist, Trello та Notion, підкреслюючи їхні обмеження щодо простоти, гнучкості та індивідуального налаштування для користувача, що спонукає до створення більш адаптованого інструменту.

Технологічна основа додатку базується на сучасних веб-технологіях, таких як React для інтерфейсу, Node.js і Firebase для серверної частини та PostgreSQL або MongoDB для зберігання даних, що забезпечує масштабованість, безпеку та простоту використання. Додаток має широкі функціональні можливості, включаючи створення завдань, фільтрацію, нагадування, аналітику та адаптивний дизайн для різних пристроїв, спрямований на підвищення продуктивності користувачів і полегшення ефективного управління часом.

Розроблено заходи щодо охорони праці та безпеки в надзвичайних ситуаціях.

ЗМІСТ

ВСТУП

Ми живемо в еру динамічного розвитку цифрових технологій управління часом і особистою ефективністю, що стає однією з ключових складових успішного професійного та особистого життя. Зі зростанням кількості щоденних справ, зустрічей, завдань і цілей усе більшої актуальності набувають інструменти, які дозволяють систематизувати й контролювати особисту діяльність. Одним із таких інструментів є веб-застосунки для моніторингу особистих щоденних задач, що поєднують функціональність, доступність та мобільність.

Потреба в гнучкому і зручному сервісі для організації повсякденних справ особливо актуальна для людей, які працюють у багатозадачному середовищі, студентів, фахівців у сфері IT та інших галузей. Більшість сучасних рішень або перевантажені зайвими функціями, або не забезпечують належного рівня персоналізації. Це створює попит на розробку веб-застосунку, який дозволятиме користувачеві ефективно створювати, відслідковувати й аналізувати виконання особистих задач у зручному інтерфейсі.

Метою даної кваліфікаційної роботи є розробка веб-застосунку для моніторингу особистих щоденних задач, який забезпечить простий механізм створення та контролю завдань, можливість перегляду статистики виконання, а також адаптацію під різні типи користувачів. У процесі розробки передбачено використання сучасних веб-технологій, зокрема React для створення клієнтської частини та Firebase для реалізації серверної логіки та бази даних.

Актуальність обраної теми зумовлена зростаючою потребою в цифрових інструментах самоорганізації, які забезпечують автономне управління часом і підвищення особистої продуктивності. Розробка такого веб-застосунку дозволяє не лише задовольнити практичні потреби користувачів, але й продемонструвати комплексні навички веб-програмування, архітектури застосунків і роботи з хмарними сервісами.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Актуальність теми

У сучасному динамічному світі, що стрімко цифровізується, дедалі важливішим стає ефективне управління часом, особливо в умовах постійного збільшення обсягу інформації та кількості щоденних справ. Багато людей, як у професійній, так і в особистій діяльності, стикаються з труднощами планування, пріоритезації та своєчасного виконання задач. У відповідь на ці виклики активно розвиваються інструменти для тайм-менеджменту, серед яких значне місце займають веб-застосунки для моніторингу особистих щоденних задач.

Веб-застосунок, що дозволяє користувачу створювати задачі, відстежувати їх виконання, формувати статистику та звіти, став невід'ємною частиною повсякденного життя багатьох користувачів. Серед ключових причин зростання популярності таких інструментів можна виділити необхідність підтримання продуктивності, боротьби з прокрастинацією, планування складних проєктів та балансування між роботою і особистим життям.

Оскільки основна частина цифрових процесів відбувається у веб-середовищі, веб-застосунки мають перевагу в порівнянні з настільними або мобільними рішеннями. Вони не потребують встановлення, працюють у будь-якому браузері та дозволяють зберігати дані на хмарних серверах, що забезпечує доступ до задач з будь-якого пристрою. Крім того, веб-застосунки легше оновлювати та масштабувати, що робить їх зручними для подальшого розвитку.

Таким чином, створення сучасного веб-застосунку для моніторингу особистих щоденних задач є актуальним завданням, яке дозволяє реалізувати низку функціональних та практичних можливостей з урахуванням потреб сучасного користувача.

1.2. Аналіз існуючих рішень

На ринку присутня велика кількість рішень для планування та моніторингу задач, серед яких є як прості додатки для щоденного використання, так і складні корпоративні системи. Серед найвідоміших можна виокремити такі інструменти, як **Todoist**, **Trello**, **Google Tasks**, **Microsoft To Do**, **Notion**, **Any.do**, **ClickUp** та інші.

Todoist є популярним планувальником, що дозволяє користувачу створювати проєкти, задачі, підзадачі, позначати їх мітками, пріоритетами, а також встановлювати дедлайни та нагадування. Він підтримує інтеграцію з календарем, поштовими сервісами та сторонніми платформами. Основним недоліком є те, що деякі корисні функції (аналітика, фільтрація, розширені нагадування) доступні лише у платній версії.

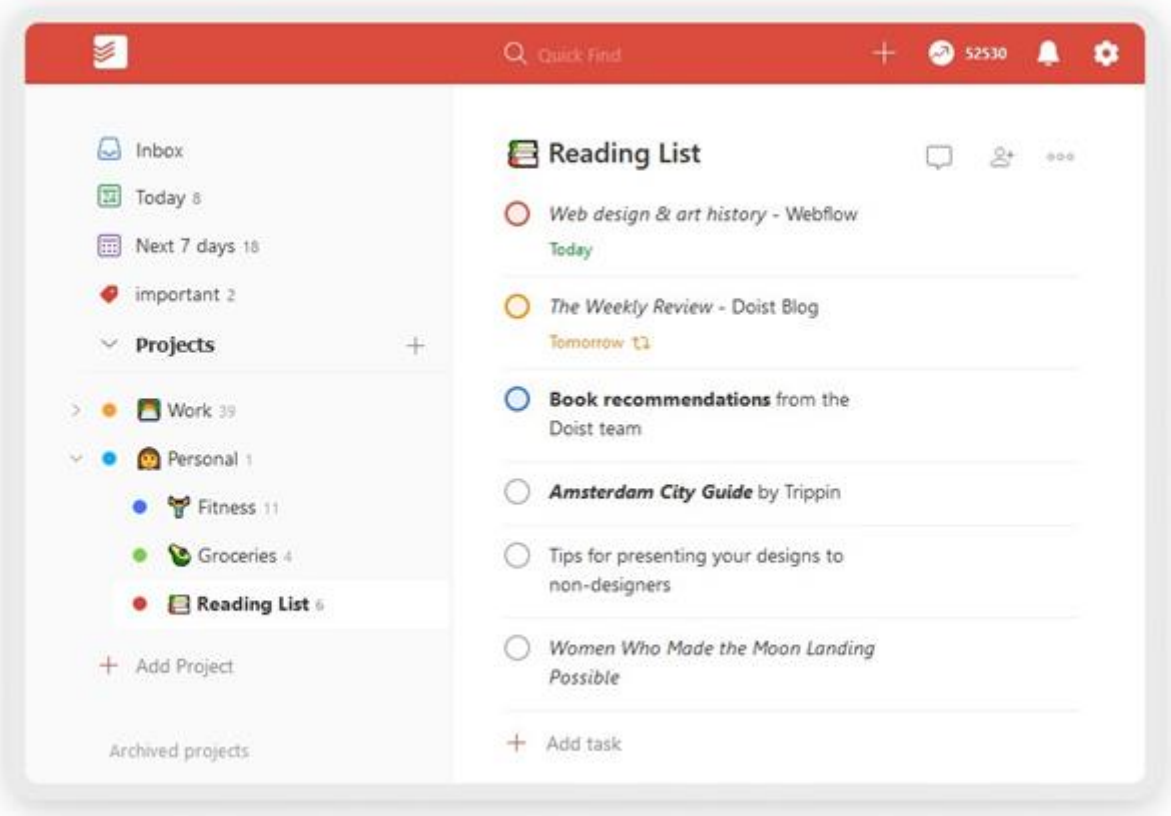


Рис. 1.1 Todoist – головна сторінка застосунку

Trello реалізує підхід Kanban, який добре підходить для візуального представлення процесу роботи. Кожна задача — це картка, яку можна переміщати між списками відповідно до статусу виконання. Trello зручний для

командної роботи та ведення проєктів, проте не надто адаптований для ведення рутинних особистих задач або трекінгу продуктивності.

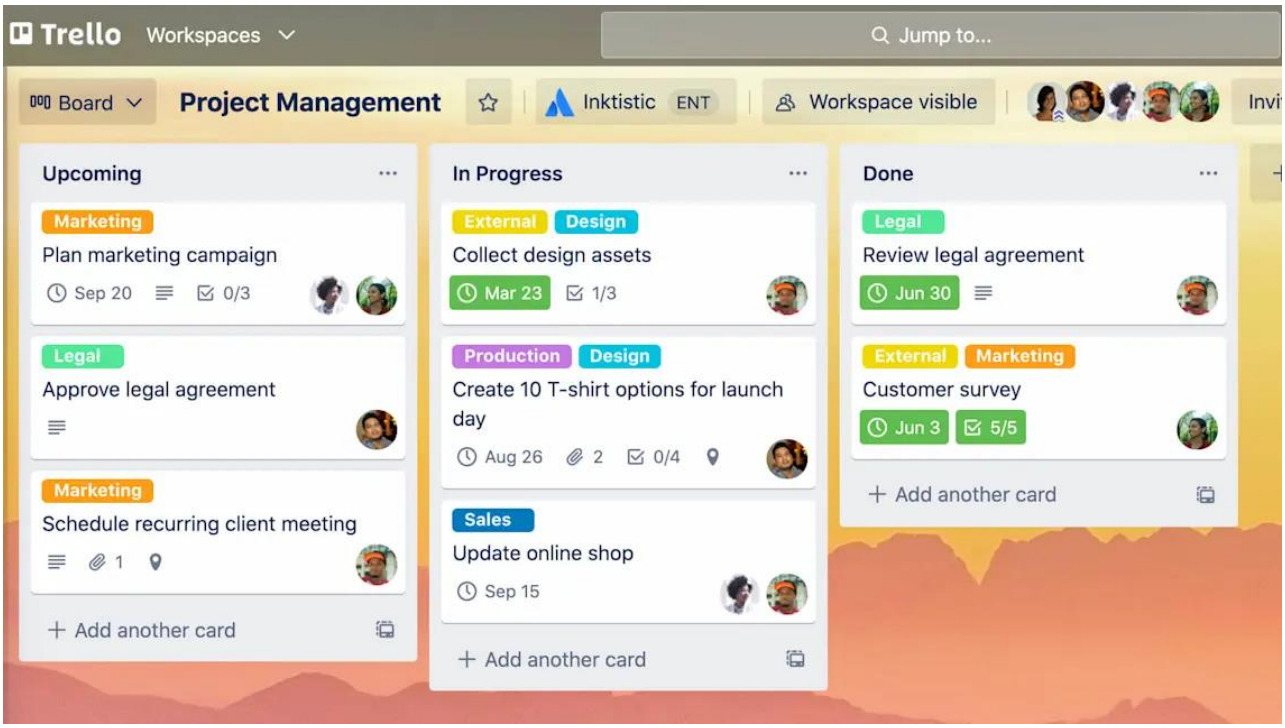


Рис. 1.2. Trello – головна сторінка застосунку

Google Tasks має простий інтерфейс і є частиною екосистеми Google, що забезпечує його тісну інтеграцію з Gmail та Google Calendar. Він дозволяє створювати списки задач і встановлювати нагадування, але має обмежений функціонал без підтримки звітності, фільтрації та пріоритезації.

Microsoft To Do — це продукт компанії Microsoft, який поступово замінив популярний сервіс Wunderlist. Його основною перевагою є глибока інтеграція з іншими продуктами Microsoft (Outlook, Teams тощо). Однак, користувачі, що не використовують екосистему Microsoft, часто стикаються з певними незручностями у налаштуванні та обмеженнях функціоналу.

Notion — багатофункціональний інструмент, який дозволяє створювати бази даних, списки, нотатки та задачі. Завдяки своїй гнучкості він може бути адаптований для потреб індивідуального користувача, проте має складний інтерфейс, що потребує певного навчання. Крім того, Notion менш зручний для щоденного моніторингу задач, якщо не реалізовано спеціалізовану структуру.

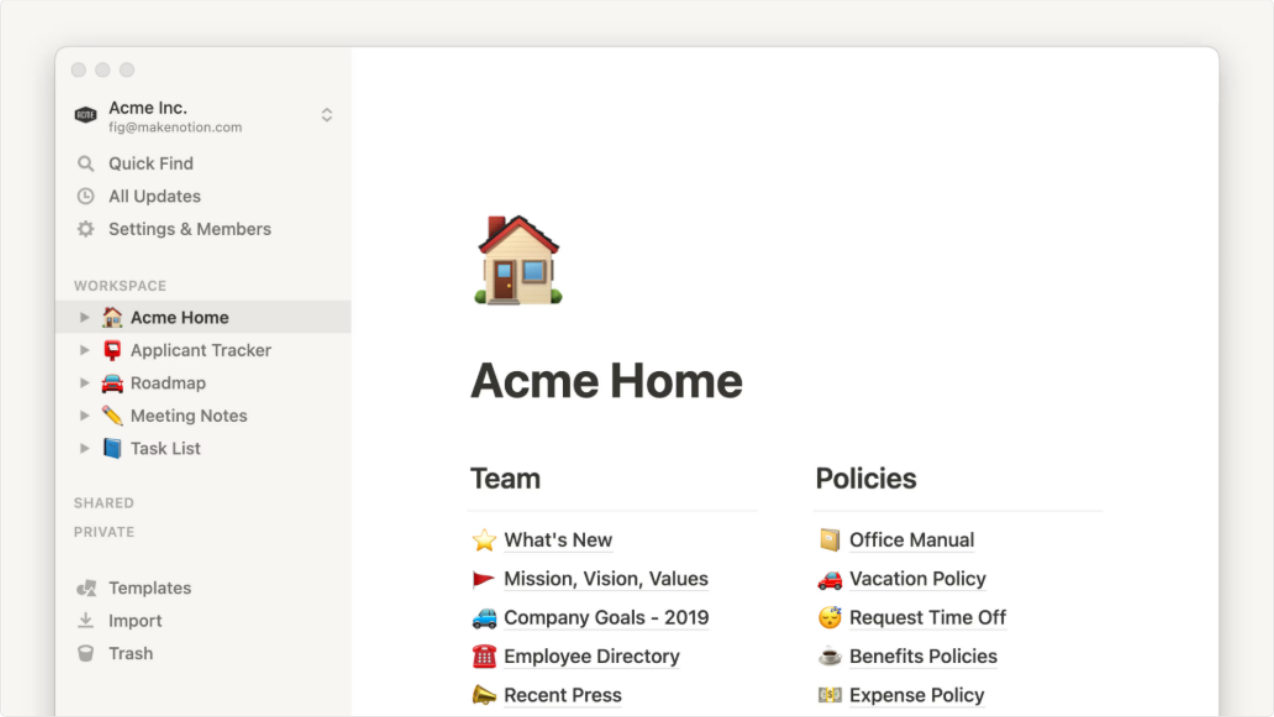


Рис. 1.3. Notion – головна сторінка застосунку

ClickUp — ще один потужний інструмент, що орієнтований на управління завданнями, командну роботу та аналітику. Він підтримує безліч функцій, включаючи тайм-трекінг, створення задач, нагадування, діаграми Ганта. Проте для індивідуального користувача його інтерфейс може виявитися занадто складним і перевантаженим.

У таблиці нижче подано короткий огляд функціональних можливостей основних сервісів:

Таблиця 1.1 Огляд функціональних можливостей основних сервісів

Сервіс	**Пріоритезація**	**Звіти**	**Нагадування**	**Інтеграція**	**Мобільність**	**Безкоштовна версія**
1	**2**	**3**	**4**	**5**	**6**	**7**
Todoist	Так	Частково	Так	Широка	Так	Обмежена
Trello	Обмежено	Ні	Так	Широка	Так	Так
Google Tasks	Ні	Ні	Так	Google	Так	Так

Продовження табл. 1.1

1	2	3	4	5	6	7
Microsoft To Do	Так	Ні	Так	Microsoft	Так	Так
Notion	Частково	Частково	Обмежено	Так	Так	Так
ClickUp	Так	Так	Так	Широка	Так	Так (з обмеженнями)

З аналізу видно, що жоден із існуючих сервісів не поєднує водночас простоти, гнучкості, звітності та повної адаптації під потреби окремого користувача. Багато рішень є або надто складними для щоденного використання, або занадто спрощеними, що не дозволяє вести глибокий моніторинг задач та ефективно аналізувати власну продуктивність.

1.3. Технологічна база проєктування веб-застосунку

У процесі розробки веб-застосунків вибір технологій має вирішальне значення, оскільки саме від цього залежить не лише функціональність, але й масштабованість, зручність користування, продуктивність, безпека, підтримка та майбутнє розширення програмного продукту. Технологічна база веб-застосунку для моніторингу особистих щоденних задач повинна відповідати сучасним стандартам, бути адаптивною до змін середовища та вимог користувача, а також забезпечувати належний рівень інтеграції між клієнтською та серверною частинами.

Клієнтська частина, або фронтенд, у контексті даного проєкту повинна забезпечувати високий рівень взаємодії з користувачем у реальному часі, з плавною реакцією інтерфейсу на дії користувача та миттєвою візуалізацією результатів. Найбільш ефективними інструментами для побудови такої частини є сучасні JavaScript-фреймворки, серед яких особливу увагу заслуговує React. Цей фреймворк створено з орієнтацією на компонентний підхід до програмування, що дозволяє створювати багаторазові модулі інтерфейсу. React має широку екосистему, активне ком’юніті розробників і численні бібліотеки, які значно полегшують реалізацію складних UI-рішень. Комбінація React з такими

бібліотеками, як Redux або Context API, дозволяє ефективно керувати станом додатку, а використання засобів маршрутизації, таких як React Router, сприяє зручному переходу між сторінками в межах SPA (Single Page Application).

Серверна частина, або бекенд, має на меті обробку запитів користувача, керування базою даних, виконання логіки застосунку, автентифікацію та авторизацію. Одним із найбільш гнучких і популярних підходів до створення бекенду є використання середовища виконання Node.js, яке дозволяє створювати швидкі, асинхронні серверні застосунки на мові JavaScript. Разом із фреймворком Express.js така технологія забезпечує гнучкість структури, ефективну маршрутизацію та зручну роботу з HTTP-запитами. Крім того, Node.js надає можливість обробляти велику кількість одночасних з’єднань, що важливо для сучасних веб-застосунків з високим навантаженням.

Важливою складовою є база даних, яка повинна зберігати інформацію про задачі, користувачів, події, метадані та інші об'єкти системи. У випадку потреби у зберіганні структурованих даних із чітко визначеними зв’язками між сутностями доцільним є використання реляційної бази даних, наприклад PostgreSQL. Ця система управління базами даних поєднує продуктивність, надійність та розширюваність, а також підтримує складні запити, транзакції та перевірку цілісності даних. Якщо ж структура даних передбачає часті зміни та більшу гнучкість у зберіганні, альтернативним варіантом є використання документоорієнтованої бази даних MongoDB, яка краще пристосована до швидкої розробки та роботи з JSON-подібними структурами.

Забезпечення безпеки в роботі з персональними даними передбачає реалізацію надійних механізмів автентифікації та авторизації. Одним із поширених підходів є використання JSON Web Token (JWT), що дозволяє створювати зашифровані токени доступу, які перевіряються при кожному запиті користувача без необхідності зберігати сесію на сервері. У контексті захисту та обліку користувачів також доцільним є впровадження протоколу OAuth 2.0 або використання платформ на зразок Firebase Authentication, які дозволяють реалізувати входи через Google, Facebook та інші сервіси.

З погляду користувацького інтерфейсу особливу увагу слід приділити адаптивному дизайну, що гарантує зручність користування застосунком на різних пристроях — від мобільних телефонів до великих моніторів. Для цього застосовуються сучасні CSS-фреймворки, серед яких слід виокремити Tailwind CSS, що надає розробникам потужний набір утилітарних класів для створення швидкого, гнучкого та мінімалістичного дизайну.

На завершення слід зазначити, що сучасні вимоги до якості програмного забезпечення передбачають впровадження практик автоматизованого тестування, безперервної інтеграції та доставки (CI/CD), а також контейнеризації середовища за допомогою Docker. Це дозволяє не лише пришвидшити процес розробки, але й мінімізувати помилки при розгортанні програмного забезпечення у продуктивному середовищі. Таким чином, вибір технологій має ключове значення для успішної реалізації веб-застосунку та забезпечення його надійної та зручної експлуатації.

1.4. Вимоги до функціоналу застосунку

Функціональні вимоги до веб-застосунку для моніторингу особистих щоденних задач визначаються насамперед на основі цільового призначення програмного забезпечення, очікувань користувачів, аналізу аналогічних рішень, а також досвіду взаємодії з подібними системами. Основна ідея полягає у створенні інструменту, який дозволяє користувачам не лише фіксувати заплановані дії, але й відстежувати їх виконання, аналізувати динаміку продуктивності, підвищувати особисту ефективність та формувати сталі звички управління часом.

Перш за все, застосунок повинен надавати користувачеві можливість створення задач із зазначенням обов’язкових і додаткових параметрів, таких як назва задачі, її короткий опис, дата і час завершення, рівень пріоритету та належність до певної категорії або теги. Крім того, має бути забезпечена підтримка різних статусів задач, що дозволить фіксувати їх поточний стан: активна, завершена, відкладена або невиконана. Це дає змогу користувачеві

легко орієнтуватися у списку справ та оперативно приймати рішення про подальші дії.

Крім базових функцій додавання та редагування задач, важливою складовою є можливість ефективного пошуку та фільтрації. Інтерфейс застосунку повинен передбачати механізми сортування задач за критеріями, такими як дата створення, дедлайн, пріоритет або статус. Це сприяє підвищенню зручності користування застосунком при роботі з великою кількістю записів, дозволяючи зосереджуватись лише на релевантній інформації.

Особливу увагу слід приділити впровадженню системи нагадувань, яка дозволяє користувачеві отримувати сповіщення про наближення або перевищення термінів виконання задач. Це може бути реалізовано як за допомогою push-повідомлень у браузері, так і шляхом інтеграції з календарем або електронною поштою. Такий підхід сприяє уникненню пропуску важливих завдань та дозволяє краще контролювати розподіл часу.

Однією з ключових функцій, що відрізняє сучасний застосунок від спрощених блокнотів або традиційних to-do листів, є реалізація можливості аналітики діяльності користувача. На основі зібраних даних про виконані та невиконані задачі, середній час виконання, кількість відкладених справ тощо застосунок може будувати діаграми та звіти, що допоможуть користувачу оцінити свою ефективність у динаміці, виявити слабкі місця у тайм-менеджменті та сформулювати нові підходи до самоорганізації.

З урахуванням широкої аудиторії потенційних користувачів застосунок має бути повністю адаптивним, з підтримкою різних пристроїв, операційних систем та роздільної здатності екранів. Важливо також забезпечити реєстрацію та автентифікацію користувачів, що дозволить зберігати індивідуальні налаштування, синхронізувати дані між пристроями та захистити персональну інформацію. Система користувачів може включати підтримку резервного копіювання даних, що підвищує надійність роботи з застосунком та дозволяє уникнути втрат у разі збою або виходу з ладу пристрою.

У більш далекій перспективі доцільним може бути впровадження таких функцій, як співпраця над задачами у групах, можливість імпорту та експорту

задач, інтеграція зі сторонніми сервісами (наприклад, календарями, месенджерами чи інструментами керування проєктами), а також використання елементів штучного інтелекту для аналізу користувацької поведінки та надання персоналізованих порад з планування. Усе це сприятиме перетворенню застосунку на повноцінну платформу для цифрового тайм-менеджменту, здатну адаптуватися до потреб як окремого користувача, так і команди.

Проведений аналіз показав, що попри наявність значної кількості сервісів для управління особистими задачами, існує потреба у створенні простого, гнучкого, персоналізованого веб-застосунку, який би поєднував в собі зручний інтерфейс, надійний моніторинг та аналітику діяльності користувача. Виявлені недоліки існуючих систем дозволяють сформувати чіткі вимоги до нового програмного рішення. Використання сучасних веб-технологій дає змогу розробити ефективний, масштабований та безпечний застосунок для щоденного користування.

РОЗДІЛ 2
ПОСТАНОВКА ЗАДАЧІ ТА ПЛАНУВАННЯ ПРОЕКТУ

2.1 Вибір засобів реалізації клієнтської частини

Розробка клієнтської частини веб-застосунку є ключовим етапом, який визначає не лише зовнішній вигляд інтерфейсу, а й рівень зручності, швидкодії та загальної привабливості системи для кінцевого користувача. Оскільки цільова аудиторія передбачає щоденне використання інтерфейсу для створення, перегляду та редагування задач, вибір технологій мав відповідати критеріям доступності, швидкодії, адаптивності та підтримки сучасних стандартів веб-розробки.

У якості основних засобів реалізації фронтенду було обрано стандартні веб-технології: HTML, CSS і JavaScript. HTML (HyperText Markup Language) використовується для створення структури веб-сторінок та формування ієрархії вмісту. Він забезпечує основу, на якій будується весь інтерфейс, і дозволяє задавати семантично правильні елементи, що покращує індексацію та доступність застосунку. CSS (Cascading Style Sheets), у свою чергу, дає змогу оформити зовнішній вигляд інтерфейсу, реалізувати адаптивну верстку, забезпечити динамічну зміну стилів у відповідь на дії користувача та покращити візуальне сприйняття системи.

JavaScript був обраний як основна мова програмування для реалізації логіки на клієнтській стороні. Його гнучкість, популярність і широка підтримка з боку браузерів дозволяють реалізовувати інтерактивну поведінку компонентів, обробку подій, валідацію форм, маніпуляції DOM та асинхронну взаємодію з сервером за допомогою fetch API або бібліотек для HTTP-запитів. JavaScript також є мовою, яка підтримується більшістю фреймворків і бібліотек, що дозволяє легко масштабувати функціональність у разі потреби.

Таким чином, використання HTML, CSS і JavaScript дозволяє створити стабільну, адаптивну та інтуїтивно зрозумілу клієнтську частину веб-застосунку,

що забезпечує необхідний рівень інтерактивності, зручності та візуальної привабливості.

2.2 Застосування React для побудови компонентної структури

Хоча стандартні засоби HTML, CSS і JavaScript є достатніми для створення простих інтерфейсів, при реалізації складних, інтерактивних та динамічних веб-застосунків постає потреба в кращій організації коду та повторному використанні компонентів. Саме з цією метою було обрано бібліотеку React, яка дозволяє створювати сучасні односторінкові застосунки (SPA) з високою продуктивністю та зрозумілою архітектурою.

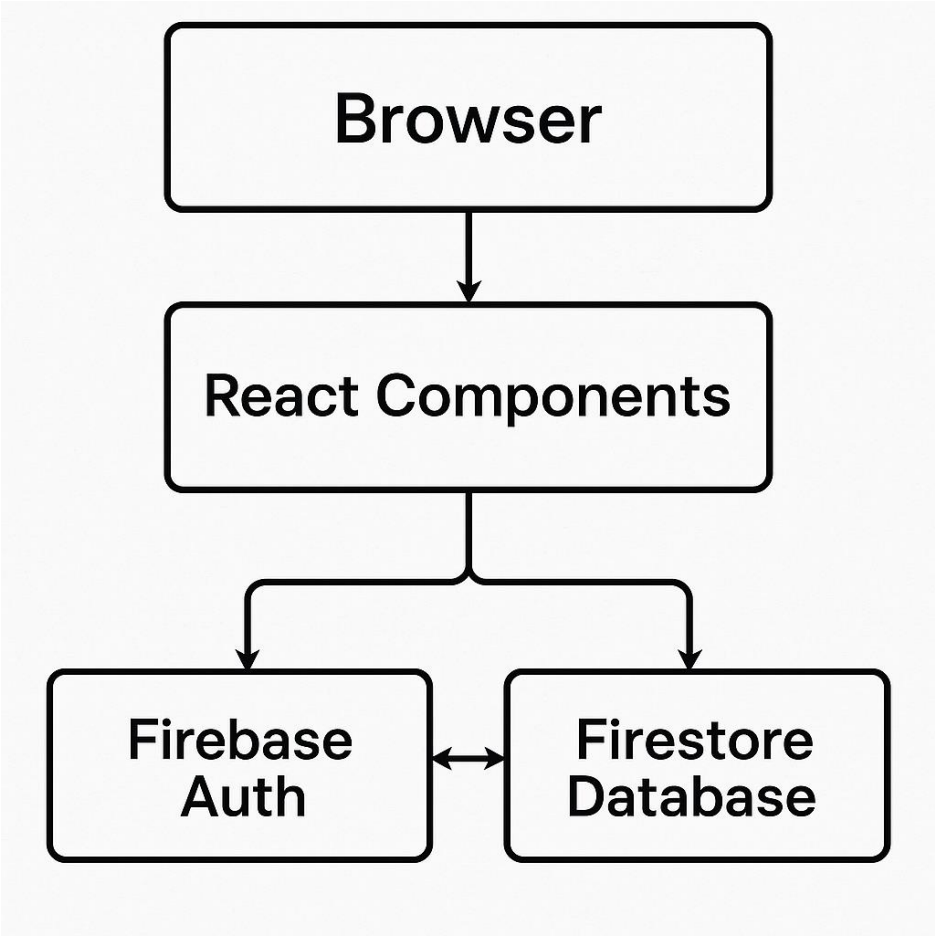


Рис. 2.1. Діаграма компонентів

React — це популярна бібліотека для побудови інтерфейсів, створена компанією Meta. Вона забезпечує компонетно-орієнтований підхід до розробки,

де кожен функціональний елемент (кнопка, форма, блок задач тощо) реалізується у вигляді незалежного модуля. Це дозволяє не лише багаторазово використовувати компоненти, а й значно спростити підтримку та розширення коду. Структуризація у вигляді дерева компонентів робить проект логічно впорядкованим та масштабованим.

Окремою перевагою React є використання віртуального DOM — абстрактного представлення реального DOM, яке дозволяє оптимізувати оновлення інтерфейсу. Завдяки цьому React не перезавантажує всю сторінку при кожній зміні, а лише оновлює ті її частини, які зазнали змін. Це суттєво покращує продуктивність, особливо на слабших пристроях.

React також підтримує JSX — синтаксис, який дозволяє поєднувати HTML-подібну розмітку з JavaScript-кодом. Це робить розробку більш зручною та природною для розробника, дозволяючи чітко бачити, як логіка взаємодіє з інтерфейсом.

У рамках реалізації проєкту з моніторингу задач React надає можливість динамічного створення і оновлення списку завдань, редагування їхніх атрибутів, фільтрації за критеріями, керування станами та реалізації зручної навігації в межах SPA. Завдяки широкій екосистемі React розширюється за допомогою додаткових бібліотек (наприклад, React Router для маршрутизації, Redux або Context API для керування станом), що дає змогу поступово збільшувати складність застосунку без кардинального переписування логіки.

Отже, застосування React дозволило реалізувати інтерфейс, який є не тільки зручним та естетичним, але й ефективним з точки зору продуктивності, структурованості та можливостей для подальшого розвитку проєкту.

2.3 Вибір платформи для бекенду та бази даних

Для реалізації серверної частини проєкту було вирішено використати платформу Firebase, яка надає повний набір хмарних сервісів для розробки, зберігання та обробки даних у режимі реального часу. Firebase є продуктом

Google, орієнтованим на розробників веб- і мобільних застосунків, що забезпечує простоту інтеграції, високу доступність та масштабованість.

Однією з ключових переваг Firebase є Firestore — NoSQL-база даних, яка підтримує зберігання даних у вигляді колекцій і документів, що дозволяє гнучко моделювати структуру завдань і пов’язаних із ними об'єктів (наприклад, мітки, категорії, статуси). База даних працює в режимі реального часу, тобто всі зміни, внесені користувачем, автоматично синхронізуються з клієнтською частиною без необхідності оновлення сторінки. Це робить застосунок більш динамічним і наближеним до досвіду роботи з нативними програмами.

Firebase також надає модулі для автентифікації, що дозволяє організувати захищений вхід у систему через email, Google-акаунт або інші провайдери. У рамках проєкту було використано email-автентифікацію, що забезпечує безпеку доступу до персональних задач кожного користувача.

Ще однією перевагою Firebase є його масштабованість: навіть у разі збільшення кількості користувачів або даних, платформа здатна забезпечити стабільну роботу без необхідності ручного адміністрування серверів. Більше того, хмарна інфраструктура Google гарантує високий рівень доступності, відмовостійкості та безпеки.

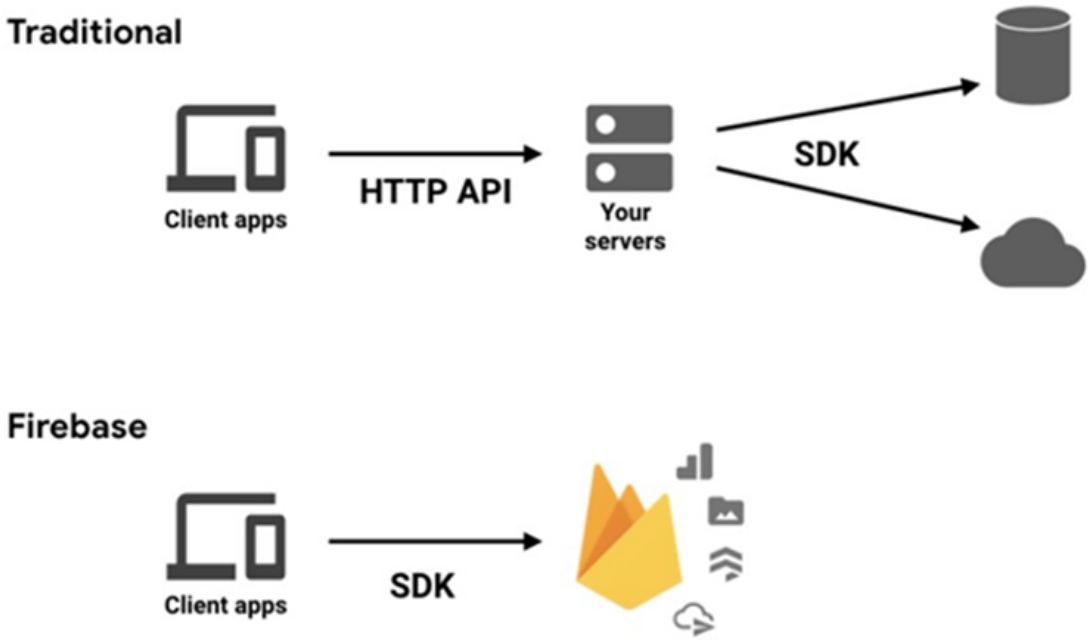


Рис. 2.2 Різниця структури сервісів з Firebase та традиційних рішень [13]

У виборі між традиційним стеком на основі Node.js із власною базою даних і Firebase, перевага була надана останньому через його спрощену інтеграцію, зменшення потреб у серверному адмініструванні, швидкий старт і зручні інструменти для відлагодження, моніторингу та розгортання.

Таким чином, використання Firebase дозволило реалізувати надійний, динамічний і масштабований серверний функціонал з мінімальними витратами на конфігурацію, що є ідеальним рішенням для застосунку, орієнтованого на широке коло користувачів і повсякденне використання.

2.4 Обґрунтування вибору середовища розробки клієнтської та серверної частини

У процесі реалізації веб-застосунку для моніторингу особистих щоденних задач вибір відповідного середовища розробки відіграє важливу роль. Від цього залежить зручність написання та відлагодження коду, інтеграція з системами контролю версій, зменшення ймовірності синтаксичних помилок та підвищення загальної ефективності роботи розробника. Саме тому підхід до вибору середовищ був ґрунтовним і враховував особливості як клієнтської, так і серверної частини системи.

Для реалізації клієнтської частини проєкту було обрано середовище розробки WebStorm (рис. 2.3), створене компанією JetBrains. WebStorm є потужним інтегрованим середовищем розробки (IDE), що спеціалізується на JavaScript, HTML, CSS та пов’язаних з ними фреймворках і бібліотеках, зокрема React, який і був використаний у даному проєкті. Однією з основних причин вибору саме цього середовища є його глибока інтеграція з React: WebStorm автоматично розпізнає JSX-синтаксис, забезпечує автодоповнення для компонентів, пропонує швидку навігацію між файлами, підказки щодо типів змінних та підтримку шаблонів коду. Це значно скорочує час розробки і знижує навантаження на розробника, дозволяючи зосередитися на реалізації логіки застосунку.

Серед переваг WebStorm також варто відзначити наявність вбудованого дебагера, що дозволяє зручно відслідковувати виконання коду у браузері, зупиняти його в критичних точках, перевіряти стан змінних і оперативно виявляти логічні помилки. Крім того, середовище підтримує інтеграцію з системами контролю версій, такими як Git, що є невід'ємною частиною сучасного програмного процесу. Це дозволило ефективно організувати контроль змін, вести історію правок та співпрацювати з іншими учасниками команди, навіть якщо проєкт реалізується індивідуально.

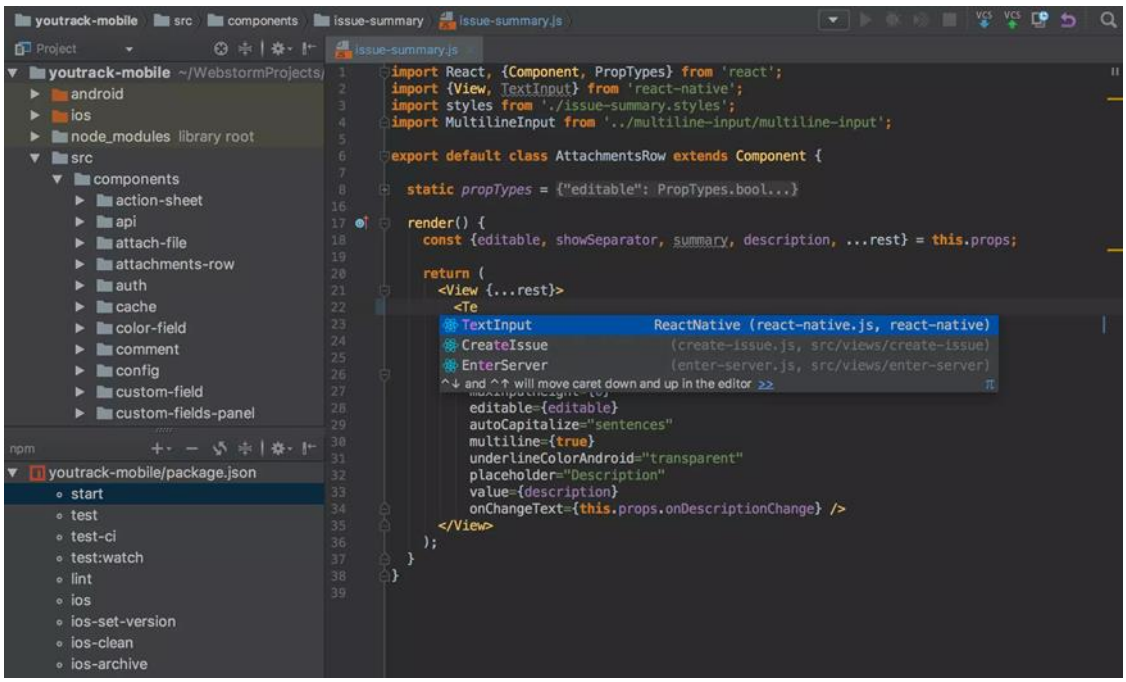


Рис. 2.3. Середовище розробки WebStorm

З точки зору роботи з бібліотеками та зовнішніми пакетами, WebStorm забезпечує тісну інтеграцію з Node.js і npm, що полегшує інсталяцію залежностей, запуск локального сервера розробки, а також керування скриптами й конфігураційними файлами. Можливість налаштувати індивідуальні шаблони, автоматизацію завдань та форматування коду ще більше підвищує ефективність процесу розробки.

Що стосується серверної частини, реалізованої на основі хмарної платформи Firebase, то потреби у використанні окремого традиційного середовища розробки фактично відсутні. Firebase надає доступ до

функціональності платформи через інтерфейс Google Firebase Console, а також інтерфейс командного рядка Firebase CLI, що інтегрується безпосередньо з WebStorm або іншим редактором коду. Завдяки цьому розробник може виконувати основні дії — такі як налаштування проєкту, деплой додатку, керування автентифікацією та базами даних — безпосередньо з редактора або браузера. У випадку потреби у створенні хмарних функцій або розширеного серверного функціоналу на Node.js, WebStorm також є цілком придатним середовищем завдяки своїй підтримці backend-розробки.

Таким чином, вибір WebStorm як середовища розробки клієнтської частини був зумовлений його потужними інструментами для роботи з JavaScript і React, а також широкими можливостями дебагінгу, управління залежностями та інтеграції з системами контролю версій. Щодо серверної частини, інтеграція Firebase із хмарними інструментами та командним рядком дозволила мінімізувати складність конфігурації середовища та забезпечити швидке розгортання застосунку. Такий вибір інструментів сприяв підвищенню ефективності розробки та якості кінцевого продукту.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ МОНІТОРИНГУ ОСОБИСТИХ ЩОДЕННИХ ЗАДАЧ

3.1 Розробка та аналіз схеми-інтерфейсу системи та сценаріїв користувачів

Коли користувачі вперше заходять на веб-сайт, вони потрапляють на домашню сторінку — цільову сторінку веб-сайту (рис. 3.1). Як показано на малюнку, праворуч є привітальна ілюстрація, праворуч розташовано вступний текст, а також кнопки входу та реєстрації.

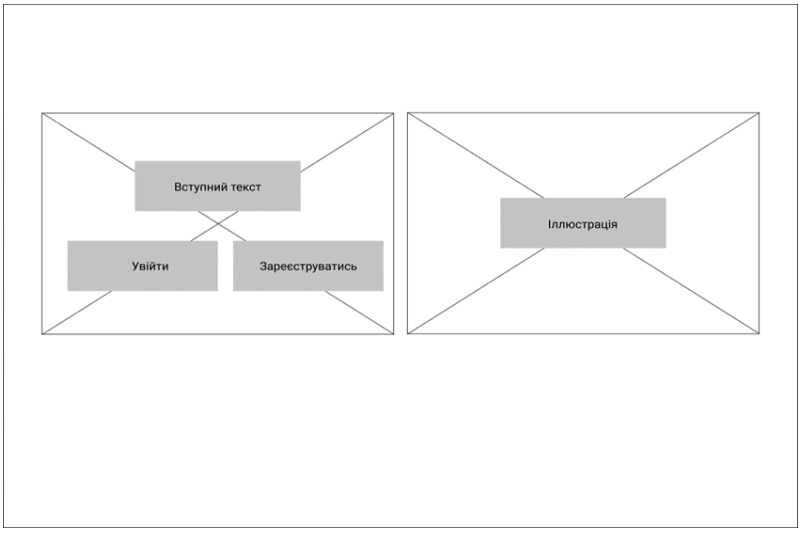


Рис. 3.1. Макет головної сторінки застосунку

Натиснувши одну з цих кнопок, користувач перейде на сторінку авторизації (рис. 3.2.). Якщо користувач ще не зареєстрований йому буде необхідно натиснути «Зареєструватися», щоб увійти на сторінку реєстрації.

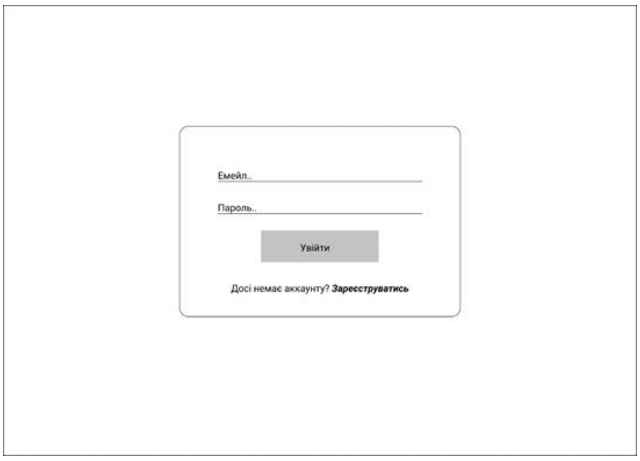


Рис. .3.2. Макет сторінки входу

Перейшовши на сторінку реєстрації, користувачі повинні пройти кроки, які складаються з трьох частин: 1 - Заповнити ім'я та прізвище (Малюнок 3.3.), 2 - вибрати категорію шаблону (Малюнок 3.4.), 3 – заповнити адресу електронної пошти та пароль (Малюнок 3.5.).

Рис. 3.3. Макет сторінки реєстрації – крок 1

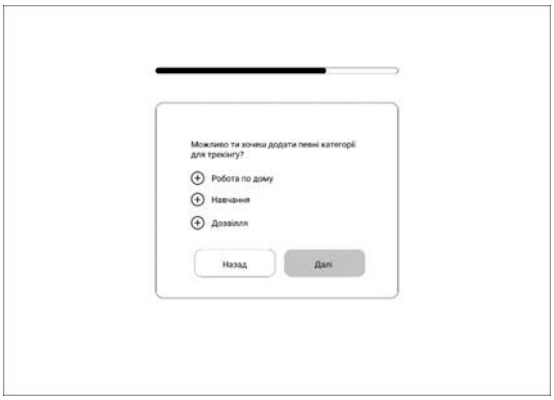


Рис. 3.4. Макет сторінки реєстрації – крок 2

Рис. 3.5 Макет сторінки реєстрації – крок 3

Після заповнення всіх необхідних полів під час входу або реєстрації користувач перенаправляється на домашню сторінку самої програми (рис. 3.6). Ліворуч знаходиться головне меню сайту, яке містить піктограми вашої сторінки користувача, домашньої сторінки, календаря та сторінки для додавання категорій або подій.

Праворуч розташована основна інформація: нагадування, завдання на сьогодні, шаблонні блоки з інформацією.

Рис. 3.6. Макет головної сторінки авторизованого користувача

Натиснувши значок профілю користувача, користувачі можуть переглянути інформацію про свій обліковий запис і статистику (Малюнок 3.7.).

Рис. 3.7. Макет вінка «Профіль користувача»

Натиснувши піктограму календаря, користувач може побачити календар із усіма запланованими завданнями (Малюнок 3.8).

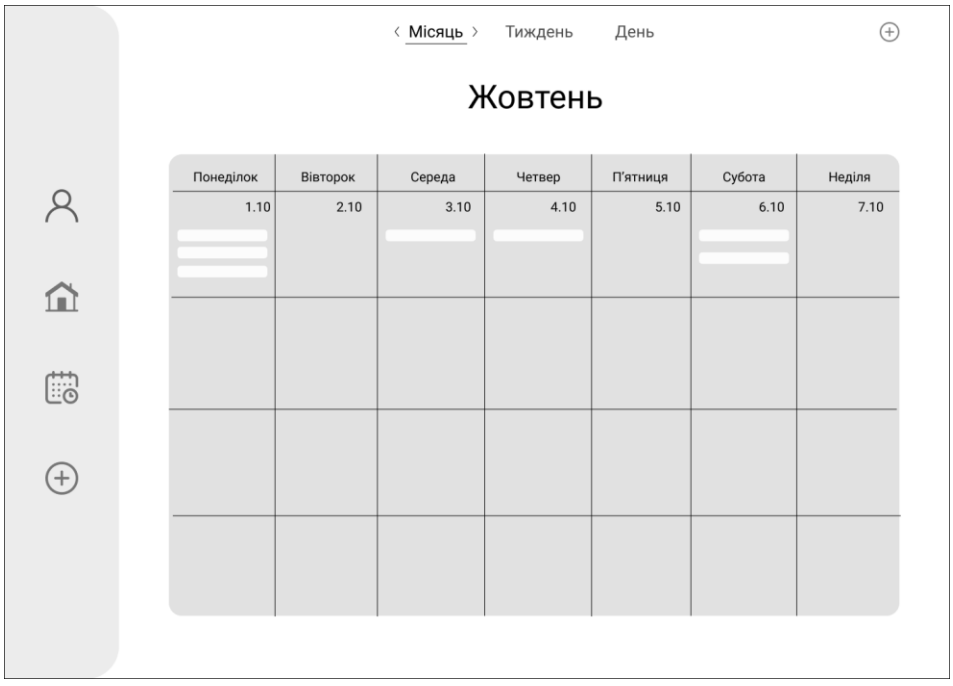


Рис. 3.8.Макет вікна «Календар користувача»

Натиснувши піктограму «Додати», користувач перенаправляється на сторінку з відповідною формою для додавання категорії чи події (рис. 3.9.).

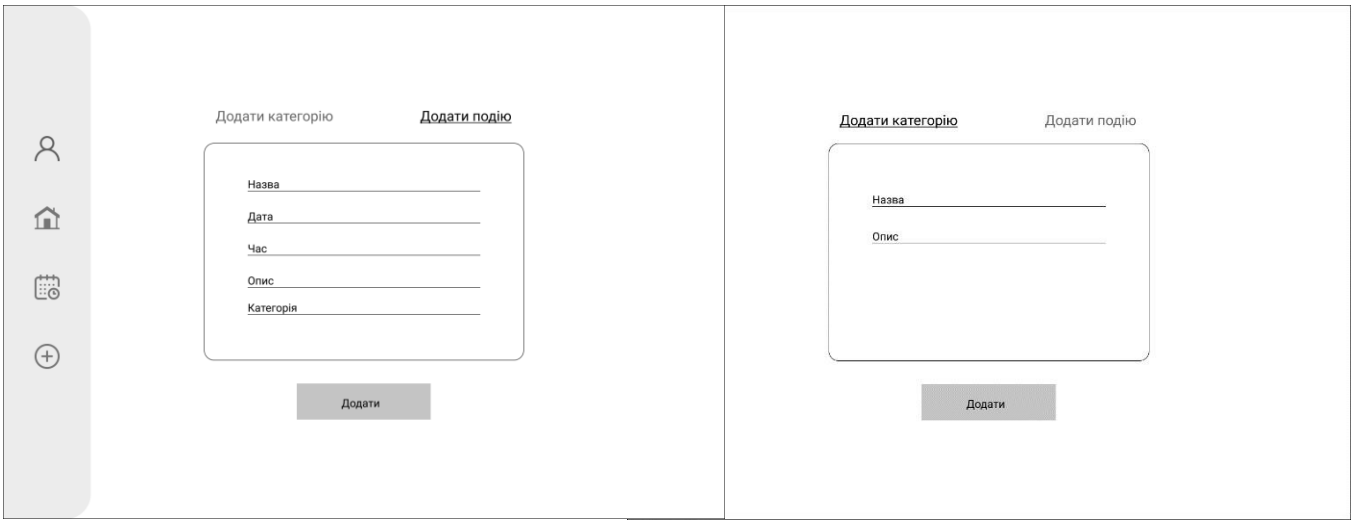


Рис. 3.9. Макет сторінки створення «нової категорії/події»

Розробляючи інтерфейс програми, можна чітко ідентифікувати сценарії користувача, що згодом допоможе розділити програму на модулі. Усі сценарії представлені у формі блок-схеми.

Сценарій авторизації показаний на малюнку 3.10.

Рис. 3.10. Алгоритм реагування на дії користувача при авторизації

Після входу користувач отримує доступ до чотирьох сторінок, кожна з

яких дозволяє виконувати певні дії.

Сценарій домашньої сторінки показано на малюнку 3.11, сценарій сторінки профілю користувача – на малюнку 3.12, сценарій сторінки календаря – на малюнку 3.13, а сценарій додавання сторінки – на малюнку 3.14.

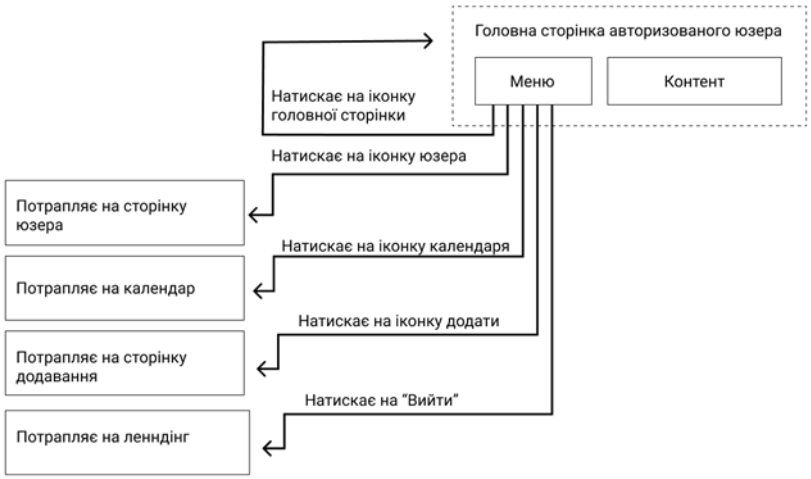


Рис. 3.11. Алгоритм реагування на дії користувача на головній сторінці

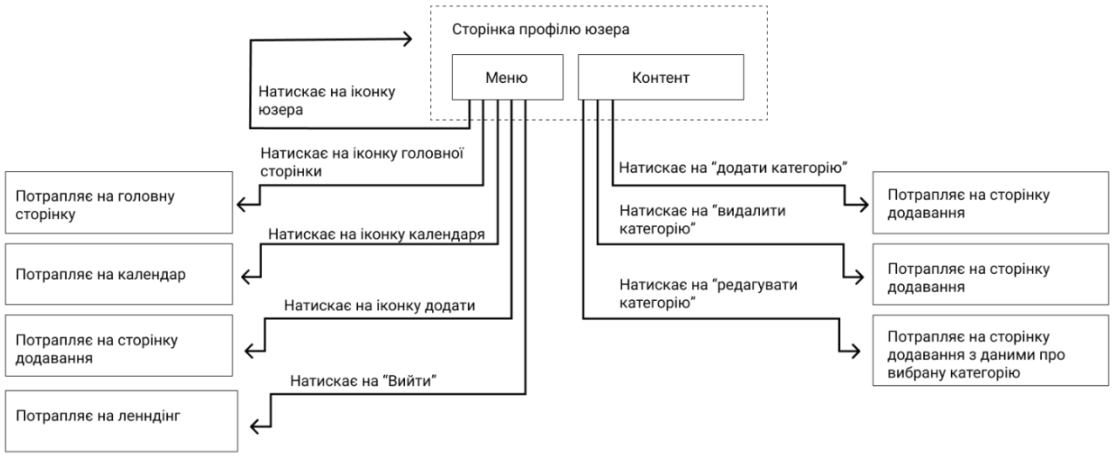


Рис. 3.12 Алгоритм реагування на дії користувача в профілі користувача

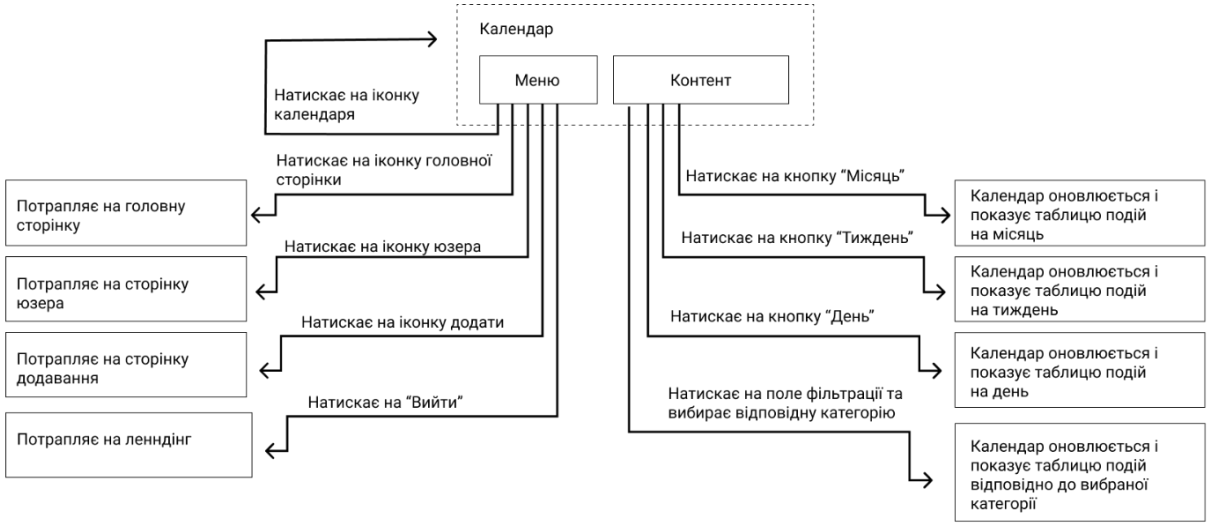


Рис. 3.13. Алгоритм реагування на дії користувача в календарі

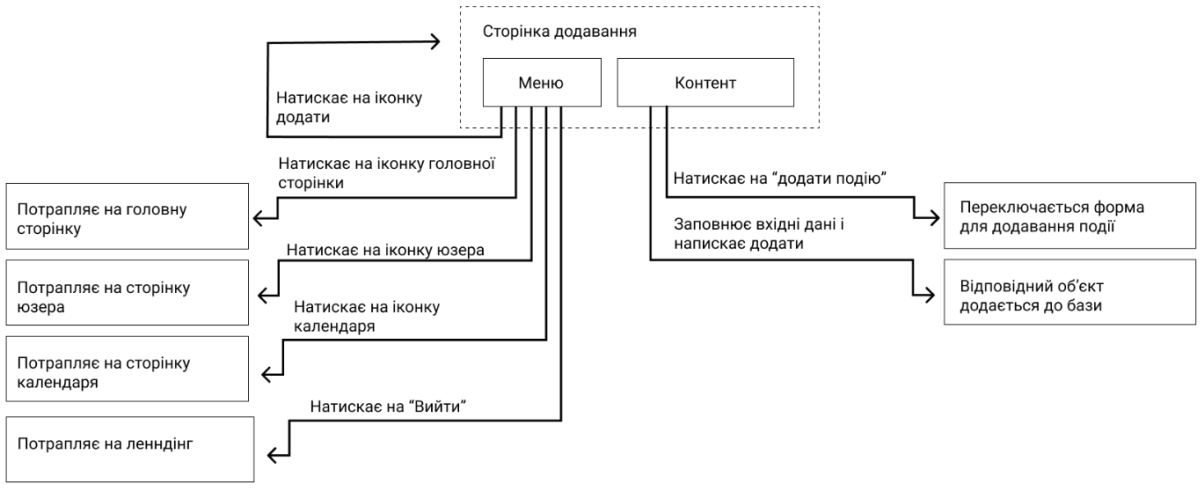


Рис. 3.14. Алгоритм реагування на дії користувача у сторінці додавання

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 користувача

5.Сторінка календаря

6.Сторінка додавання

3.2 Опис етапів розробки додатку планування особистого часу

Розробка додатку здійснювалась за наступним переліком етапів:

1. Розбивши структуру додатку на на окремі частини, отримуємо шість різних модулів: Авторизація та автентифікація; Меню користувача; Домашня сторінка; Сторінка профілю користувача; Сторінка календаря; Додати сторінку.

2. Визначено для кожного модуля необхідні вхідні дані, їх типи та обмеження: Авторизація та автентифікація. Щоб увійти в систему, введіть рядок електронної пошти та пароль.

- ім'я користувача (обов'язковий рядок)
- прізвище користувача (обов'язковий рядок)
- вибрані категорії шаблонів (можливий масив)
- електронна адреса користувача (обов'язковий рядок)
- пароль (обов'язковий рядок)

Для форми «Додати категорію» обов’язковим для заповнення є поле назви категорії. Також є додаткове поле опису категорії.

Для форми «Додати подію» обов’язковими полями є поле назви події – введіть рядок, дата події – і категорія події (вибір категорії серед категорій цього користувача) – введіть рядок. Також додаткове поле для опису події - типу string, і чекбокс для перевірки терміновості цієї події - типу boolean.

Зібравши всі вхідні дані, ми розбиваємо їх на сутності та отримуємо 4 таблиці: користувачі, категорії, події та цитати. На малюнку 3.15 схематично зображено структуру кожної таблиці.

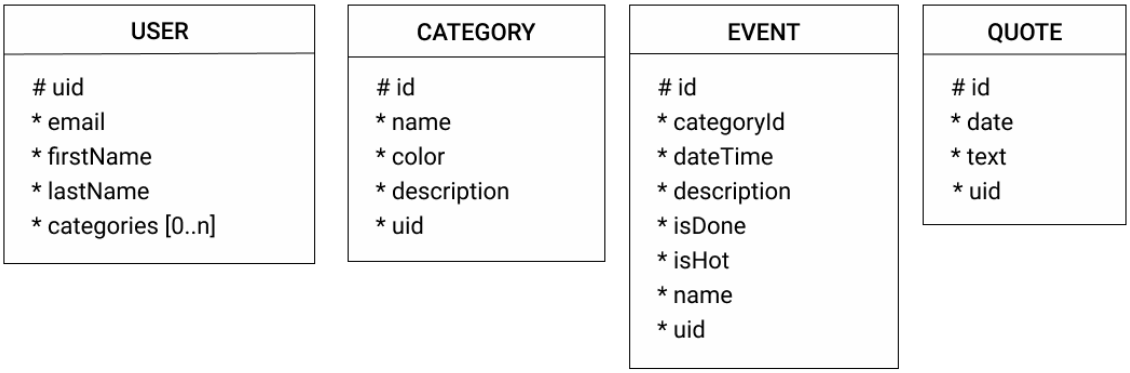


Рис. 3.15 Візуалізація представлення сутностей в базі даних.

Для підключення бази даних Firebase до клієнтської частини розробленого застосунку було створено файл firebase.js, у якому було підключення до сховища.

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

const firebaseConfig = {
    apiKey: "-",
    authDomain: "personal-tracker-c0d5b.firebaseapp.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;
```

Для ефективного використання даних користувача, підвищення продуктивності програми та уникнення повторних запитів до сервера використовується сховище Redux. Для цього під час розробки створюється окрема папка для зберігання, де зберігаються всі дії та редуктори для репозиторію. Сама конфігурація 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))
)
```

Під час першого запуску програми у файлі App.js виконуються всі необхідні запити до сервера. У цьому файлі зроблено такі запити:

Запит на отримання всіх завдань користувача: метод отримання всіх завдань користувача реалізовано як спеціальний хук useTasks, який приймає ідентифікатор користувача як вхідні дані та отримує всі події useTasks для цього користувача на виході:

```
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 => ({
```

Запит на отримання всіх цитат для певного користувача: метод отримання всіх цитат для певного користувача реалізовано як спеціальний хук useUserQuotes, який приймає ідентифікатор користувача як вхідні дані та отримує всі цитати для цього користувача як вихідні дані.

```
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 категорій шаблонів: метод отримання всіх категорій шаблонів реалізовано як спеціальний хук useTemplateCategories, який не приймає жодних вхідних даних, а на виході ми отримуємо всі категорії шаблонів.

Запит на отримання інформації про користувача: метод отримання всієї інформації про користувача реалізовано як спеціальний хук useUserInfo, який приймає ідентифікатор користувача як вхідні дані та повертає необхідну інформацію.

```
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};
}
```

Усі дані з сервера зберігаються в репозиторії, щоб їх можна було легко

повторно використовувати на будь-якій іншій сторінці.

Щоб реалізувати вхід, ми створюємо метод, який надсилає запит на сервер, перевіряє вхідні дані та повертає відповідь. Спосіб 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;
        }
    })
```

Під час реалізації процесу 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;
        }
    })
```

Реалізація функціоналу домашньої сторінки користувача. Для домашньої сторінки нам потрібно отримати нагадування користувача, прогнози та завдання на сьогодні.

Щоб отримати події, які можуть попередити користувача, ми фільтруємо за датою події, яка є найближчим днем, але не сьогодні. В результаті отримуємо такий спосіб:

```
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');
}
```

Щоб знайти всі події на сьогодні, ми фільтруємо за датою події, яка є сьогодні. Отримуємо такий результат:

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

Відсоток виконаних завдань на сьогоднішній день розраховується за таким методом:

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

Реалізація функціоналу сторінки профіля користувача. Для сторінки профілю вам потрібно отримати всі категорії для цього користувача та статистику щодо його активності.

Щоб отримати категорії, ми використали звичайну фільтрацію на основі інформації про користувачів, яка вже є в репозиторії.

Існуючі дані також були відфільтровані для отримання статистики користувачів. Частина візуалізації графа подій користувача реалізована за

допомогою бібліотеки Nivo [14].

Реалізація функціоналу сторінки додавання. Додавання сторінки категорії не вимагає завчасного отримання будь-яких даних із бази даних. Для сторінки додавання події вам потрібно отримати всі категорії для цього користувача, щоб мати змогу реалізувати вибір у формі зі значень лише з цих категорій. Додайте категорії за допомогою методу fireAddCategory 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, реалізація якого подана нище:

```
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);
        });
```

Реалізація функціоналу сторінки календаря. Для сторінки календаря вам потрібно отримати всі категорії та всі їхні події для певного користувача зі

сховища. Фільтрування подій здійснюється за допомогою методу changeEventCategory, як показано нижче:

```
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))
    }
}
```

Візуалізаційна частина календаря реалізована за допомогою бібліотеки Fullcalendar [15].

3.3. Створення об'єктів і опис інтерфейсу основної програми

Програма дозволяє створювати користувачів, створювати категорії та створювати певні події під час авторизації чи автентифікації.

Коли користувача створюється, сервер надсилає нам його маркер, який потім зберігається в localStorage і діє як ідентифікатор користувача.

Коли створюється нова категорія або подія, клієнт надсилає всі необхідні дані для створення нового документа, який автоматично створюється та додається до бази даних.

Остаточний інтерфейс базується на діаграмах інтерфейсу, розроблених у розділі 3.1, оскільки вони є розробленими прототипами.

Коли користувач відкриває додаток, він переходить на головну сторінку, як показано на малюнку 24.

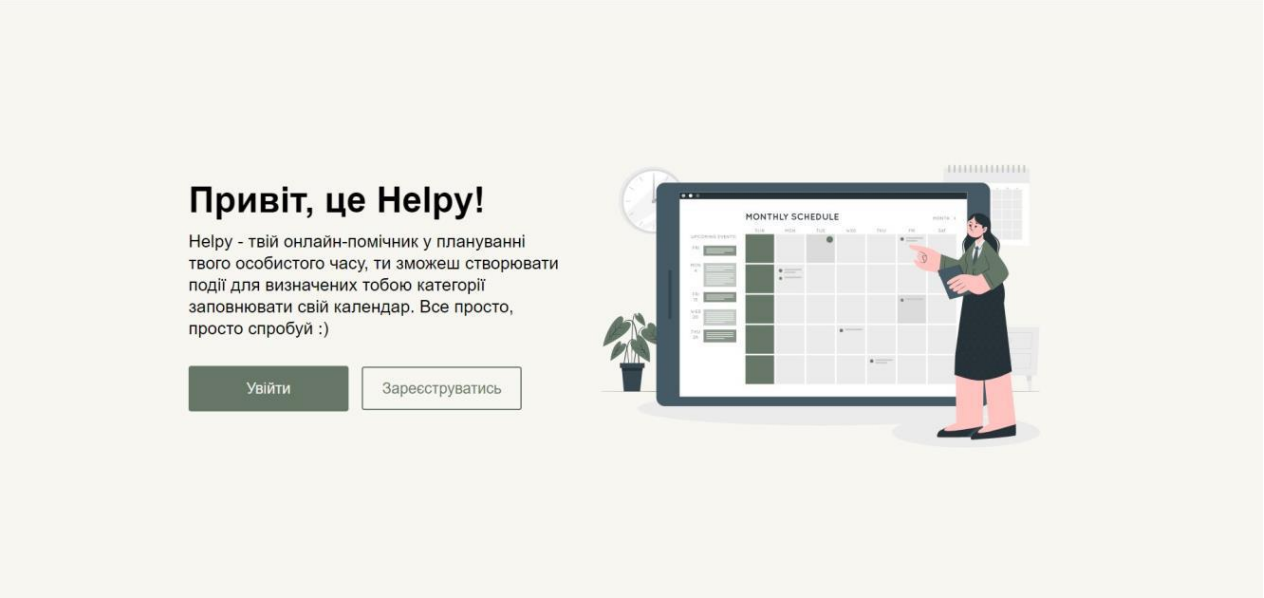


Рис. 3.16 . Основна сторінка розробленого додатку

Після цього користувачі можуть увійти або зареєструватися, натиснувши відповідну кнопку. Після натискання кнопки «Увійти» користувач побачить форму, показану на малюнку 3.17. Вибираючи реєстрацію, користувачеві необхідно заповнити три послідовні форми, показані на малюнках 3.19, 3.20 і 3.21.

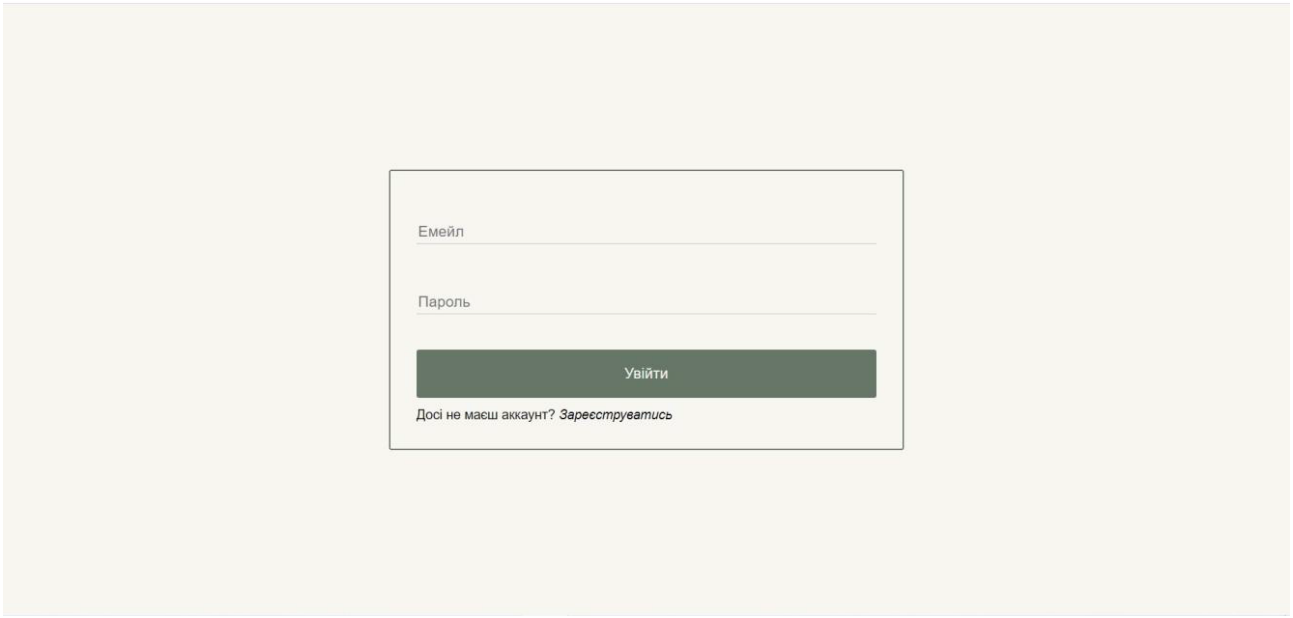


Рис. 3.17 Головна сторінка входу в додаток

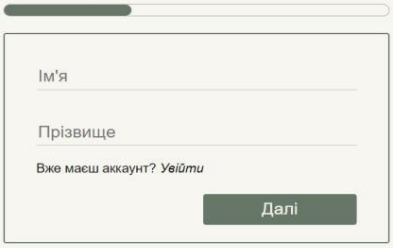


Рис. 3.18. Сторінка реєстрації (перший крок).

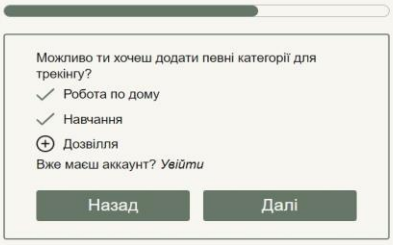


Рис. 3.19. Сторінка реєстрації (другий крок).

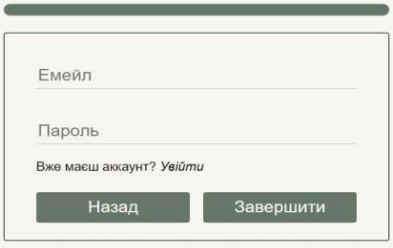


Рис. 3.20. Сторінка реєстрації (заключний крок).

Після реєстрації користувач побачить головну сторінку, як показано на малюнку 3.21.

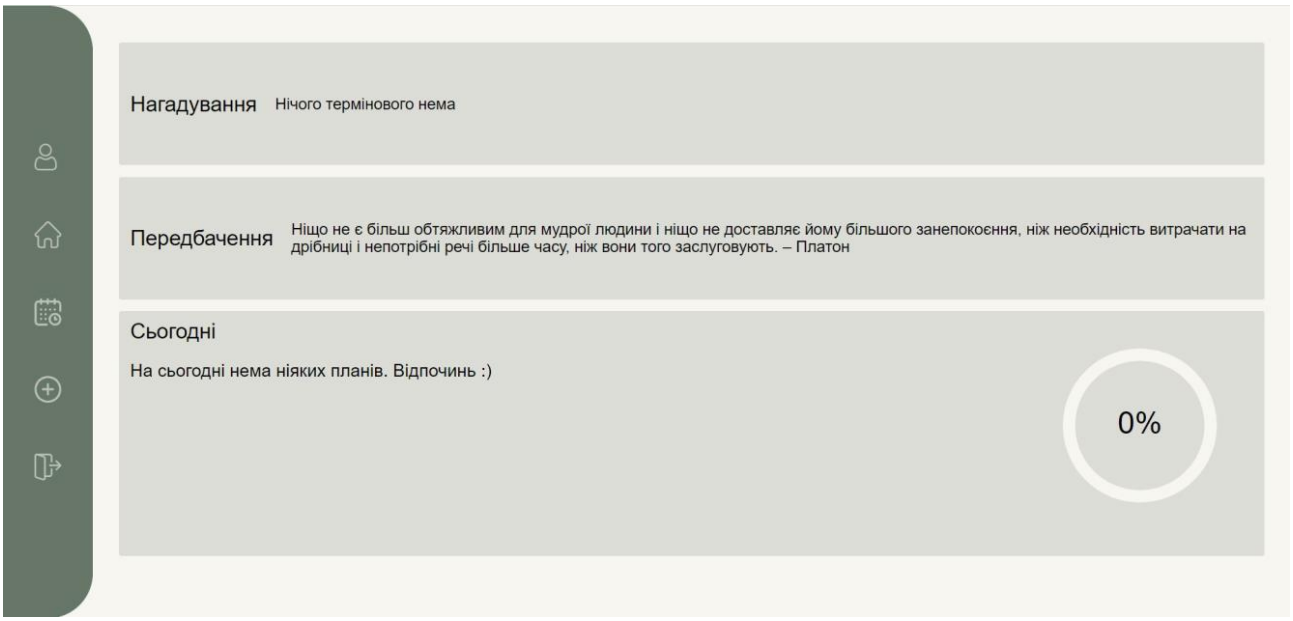


Рис. 3.21. Головна сторінка авторизованого користувача.

Якщо користувач вже користується сервісом, тобто має певні дані, і авторизується, він також побачить домашню сторінку, але заповнену необхідними даними, що можна побачити на малюнку 3.22:

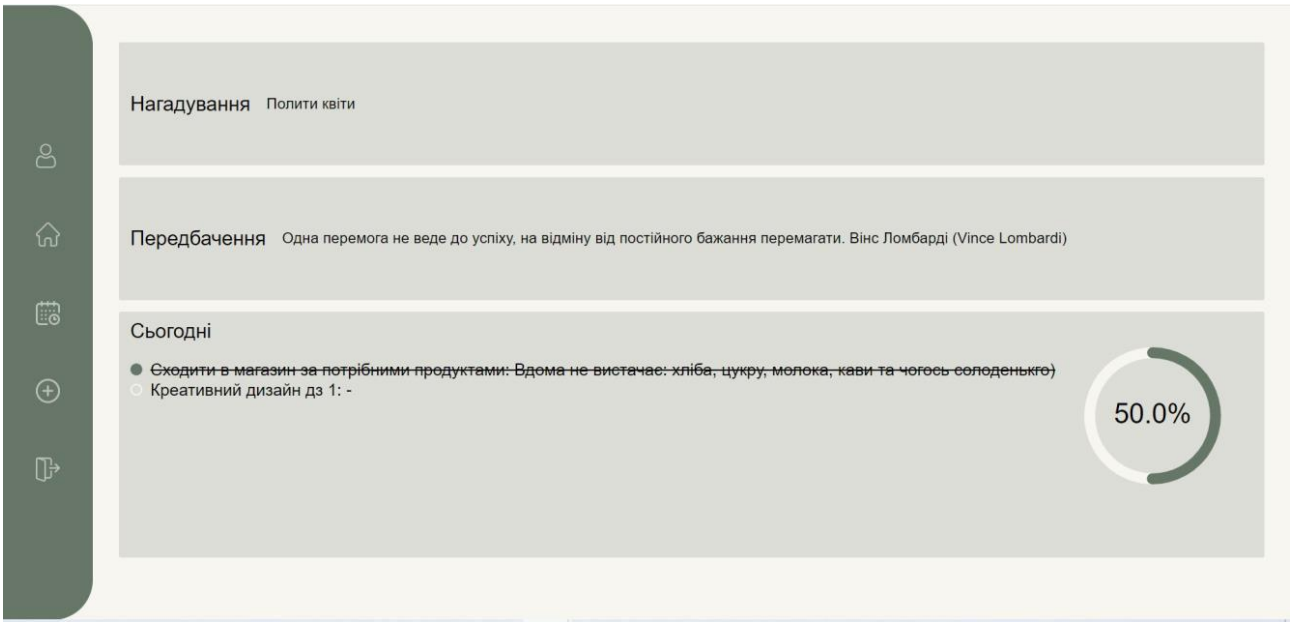


Рис. 3.22 Головна сторінка користувача з даними

Якщо користувач клацне піктограму календаря, він побачить сторінку календаря, заповнену всіма подіями користувача. Якщо користувач хоче бачити події лише в певній категорії календаря, він може вибрати відповідну категорію вгорі, і події в календарі автоматично оновляться, як показано на малюнку 3.23:

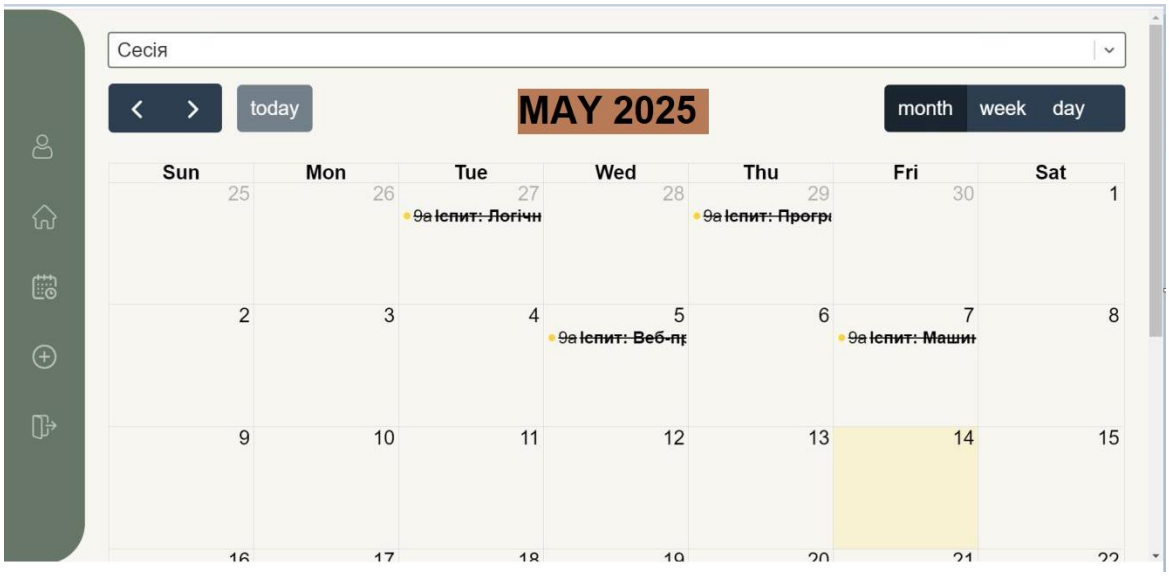


Рис. 3.23. Сторінка календаря користувача при активному фільтрі категорії «Сесія»

Якщо користувач натисне піктограму «Додати», він перейде на сторінку додавання категорії або події. Форми додавання категорії та події показано на рисунку 3.24 та 3.33 відповідно.

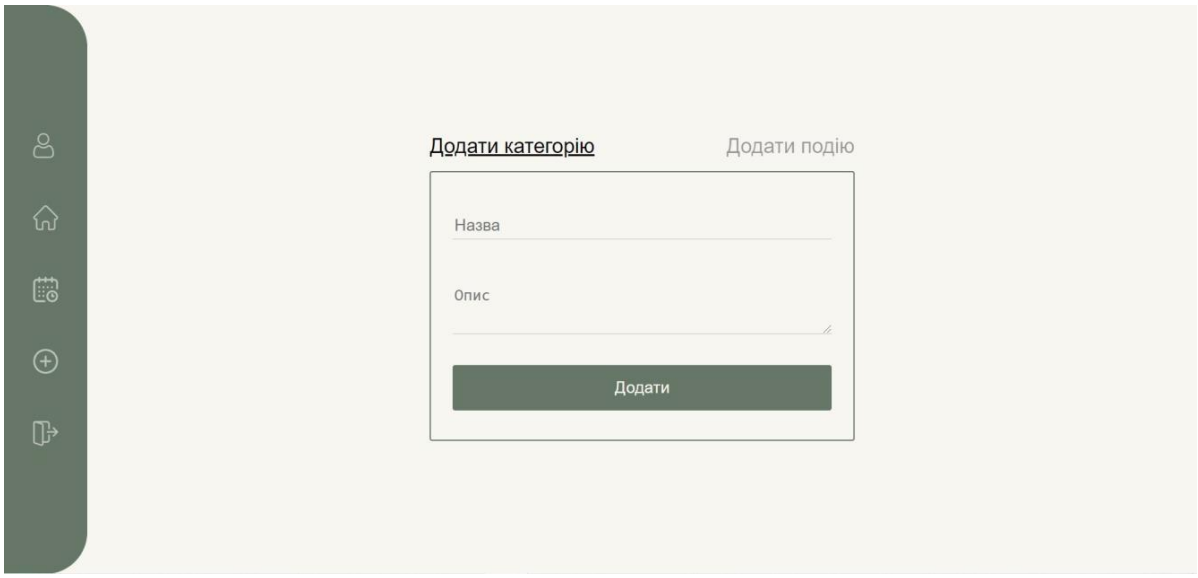


Рис. 3.24. Форма «Додати категорію»

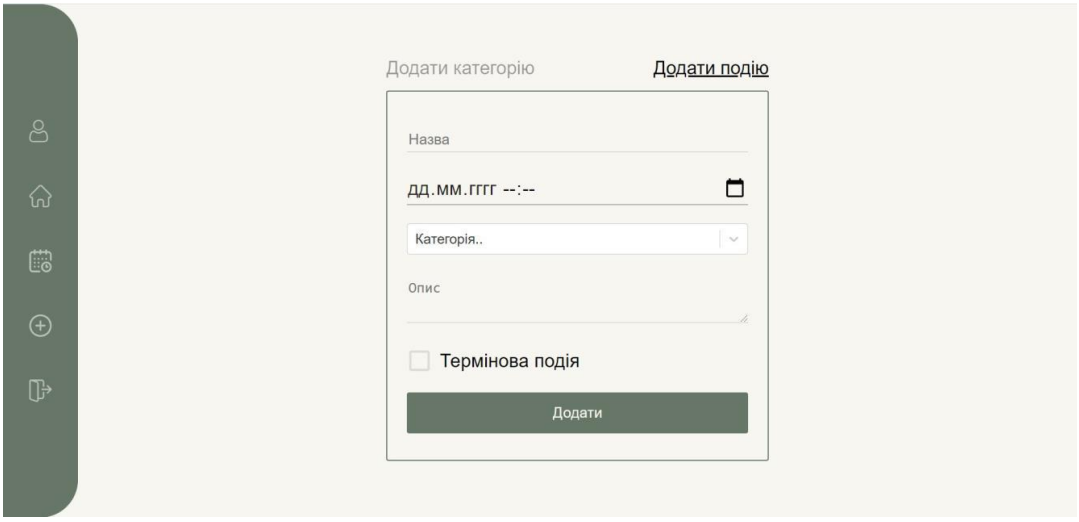


Рис. 3.25. Форма «Додати подію»

Нижче наведено приклади інтерфейсу, розробленого для мобільних пристроїв у мобільній версії:

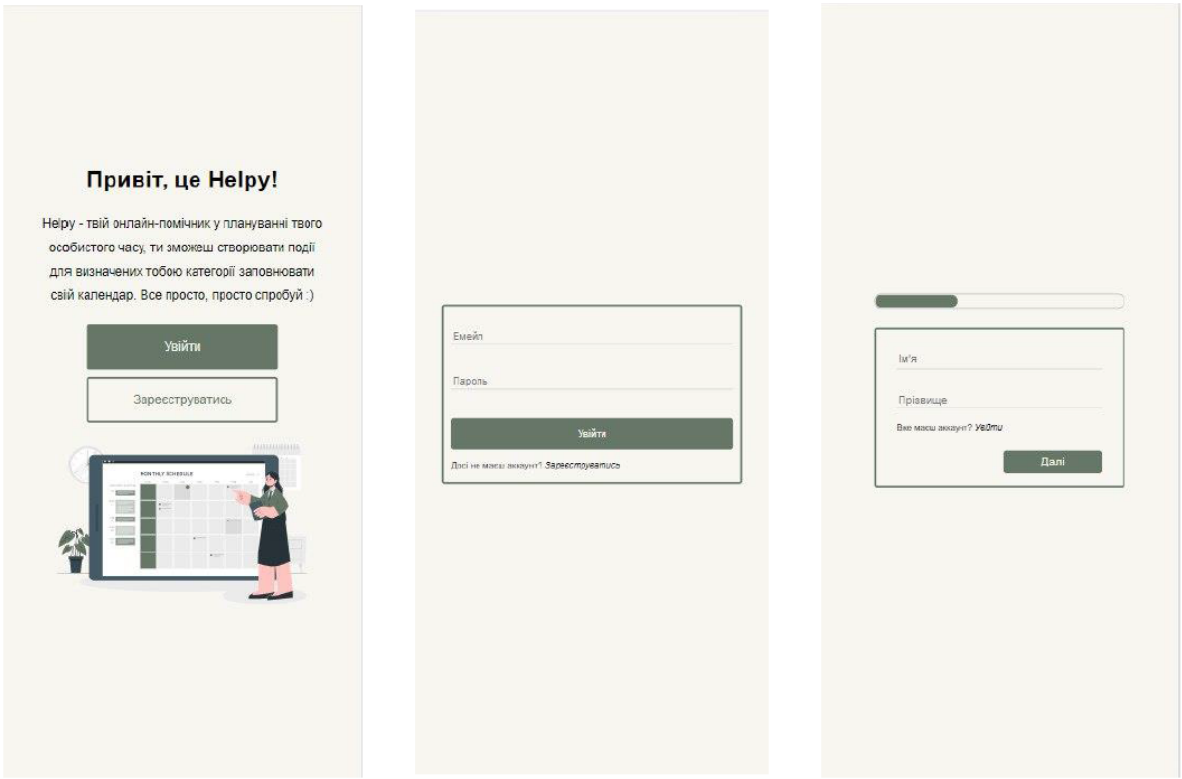


Рис. 3.26. Приклади інтерфейсу для мобільних пристроїв.

Як бачимо, створений нами застосунок, чудово адаптований і для мобільних пристроїв, що дозволить користувачам більш інтенсивно і продуктивно ним користуватися для планування особистих задач.

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Структурно-функціональний аналіз виробничого процесу та розроблення моделі травмонебезпечних ситуацій

У зображеннях процесів формування, виникнення аварій та виробничих травм усі випадкові події, що утворюють конкретну аварійну ситуацію, пов'язані між собою причинно-наслідковими зв'язками.

Метод логічного моделювання потенційних аварій, травм та катастроф відкриває можливість розробити досконалу систему управління ОП виробництва, яка базується на оперативному пошуку виробничих небезпек, їх глибокому аналізі й терміновому прийнятті заходів для усунення потенційних небезпек ще до виникнення травмонебезпечних та катастрофічних ситуацій. Деякі небезпечні ситуації в табл. 4.1.

Працівники, що обслуговують електрообладнання вениляційної системи, зобов'язані знати Правила безпечної експлуатації електроустановок споживачів відповідно до займаної посади або роботи, як вони виконують, і мати відповідну групу з електробезпеки [16,17].

Працівники, що порушили вимоги Правил безпечної експлуатації електроустановок, усуваються від роботи і несуть відповідальність (дисциплінарну, адміністративну, кримінальну) згідно з чинним законодавством. Такі працівники не допускаються до робіт в електроустановках без позачергової перевірки знань вимог правил безпечної експлуатації електроустановок.

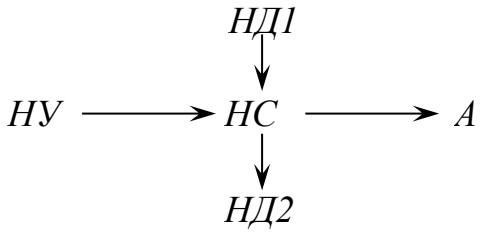
Забороняється допускати до роботи в електроустановках осіб, які не пройшли навчання і перевірку знань Правил безпечної експлуатації електроустановок.

Працівнику, який пройшов перевірку знань Правил безпечної експлуатації електроустановок, видається посвідчення встановленої форми.

Таблиця 4.1. Моделювання процесів формування та виникнення травмонебезпечних і аварійних ситуацій

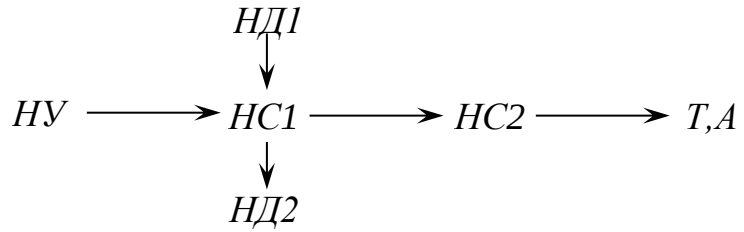
Вид робіт	Виробнича небезпека			Можливі наслідки	Заходи запобігання небезпечним ситуаціям
	Небезпечна умова (НУ)	Небезпечна дія (НД)	Небезпечна ситуація (НС)		
Використання механічної вентиляції	Оператор не перевірив обладнання НУ	Пошкоджений трубопровід мережі НД1 Закупорений трубопровід шланга НД2	Відмова вентиляційної системи (двигуна) НС	Аварія	Розвісити плакати, провести інструктажі із експлуатації обладнання системи

Модель процесу:



Використання електронних пристроїв регулювання	Пошкоджена ізоляція провідників з’єднання НУ	Пробій на корпус НД1 Коротке замикання НД2	Ураження людини електричним струмом НС1 Виведення обладнання із ладу НС2	Травма Аварія	Заміна провідників, установлення захисного обладнання (запобіжників, захист від ураження людини струмом) тощо

Модель процесу:



Посвідчення про перевірку знань працівника є документом, який засвідчує право на самостійну роботу в електроустановках на зазначеній посаді за фахом.

4.2. Вимоги техніки безпеки під час роботи обладнання та протипожежні заходи

Вимоги правил техніки безпеки перед початком роботи. Для початку роботи пов’язаної з вентиляцією вимикають рубильники або автоматичні вимикачі щита низької напруги, запирають шафу і вивішують попереджувальні плакати. Також повинні бути основні захисні засоби до яких належать такі, ізоляція яких надійно захищає від робочої напруги мережі і за допомогою яких можна дотикатися до струмопровідних частин, що перебувають під напругою, без небезпеки ураження електричним струмом (інструмент з ізольованими ручками, ізолюючі струмовимірювальні кліщі, діелектричні рукавиці).

Вимоги правил техніки безпеки під час роботи. Виконавши ці операції, надівають діелектричні рукавиці і за допомогою покажчика напруги перевіряють відсутність напруги на всіх фазах. Потім, приєднавши один кінець переносного заземлення до заземлюючого пристрою, накладають його на струмоведучі частини. Після цього остаточно приступають до роботи.

Вимоги правил техніки після закінчення роботи. Після закінчення роботи системи перед її вимиканням необхідно виконати такі технічні операції: перевірити надійність кріплення, зняти переносні тимчасові заземлення, відімкнути щит низької напруги і зняти плакати з техніки безпеки; якщо тимчасове переносне заземлення встановлене на лінії, його також треба зняти тощо [16].

Протипожежні заходи на об’єкті. Для запобігання пожеж на об’єкті розроблено організаційні, експлуатаційні, технічні режимного характеру, пожежно-евакуаційні, профілактичні заходи. До організаційних заходів відносяться правила розміщення машин, що обслуговують приміщення,

обладнання, матеріалів з дотримання певних проходів, не допускається захаращення приміщень, проходів і т.д.

4.3. Розрахунок штучного заземлення

Вибір штучного заземлення проводиться в залежності від характеру ґрунту і способу забивання стержнів [16 ,]. Розраховуємо заземлюючий контур підстанції напругою 10/0,4 кВ з глухозаземленою нейтраллю. Характер ґрунту – чорнозем з $\rho=2\cdot10^4$ Ом·см. Кліматична зона – IV (K_c – 1,2, K_n – 1,5). Струм замикання на землю в мережі становить 50 А.

В відповідності з діючими правилами, опір заземлюючого пристрою повинен становити

$$R = \frac{125}{I_з} = \frac{125}{50} = 2,5 Ом , \qquad (4.1)$$

де $I_з$ – струм замикання на землю, А.

Приймаємо 3 Ом. Контур заземлення розміщуємо в ряд з а = 5 м, l = 2,5 м. В якості стержневого заземлювача приймаємо кутникові сталь 50х50х5 мм, а протяжного – пластинчасту сталь 40х4 мм.

Опір одиночного стержня становить:

$$R_o = 0.00318\rho \cdot K_c, Ом \qquad (4.2)$$

де K_c – коефіцієнт сезонності для стержневого заземлювача (K_c – 1,2).

$$R_o = 0.00318 \cdot 2 \cdot 10^4 \cdot 1.2 = 76.32 Ом .$$

Число стержнів приймаємо 15. При цьому коефіцієнт використання стержневих заземлювачів становить η_c = 0,7. Опір всіх стержнів розтікання струму становить:

$$R_c = \frac{R_o}{n \cdot \eta_c}, Ом , \qquad (4.3)$$

де n – число стержнів, шт.

$$Rc = \frac{76.32}{15 \cdot 0.7} = 7.3Ом .$$

Довжина протяжного заземлювача становить l = 35 м (3500 см); приймаємо t = 50 см, b = 0,4 см. Опір протяжного заземлювача становить:

$$R_{пр} = \frac{0,366}{l} \cdot \rho \cdot 2 \cdot \lg \frac{2 \cdot l^2}{t \cdot b}, Ом \qquad (4.4)$$

$$R_{пр} = \frac{0,366}{3500} \cdot 1,2 \cdot 10^4 \cdot 2 \cdot \lg \frac{2 \cdot 3500^2}{0,4 \cdot 50} = 3,2Ом$$

Коефіцієнт використання протяжного заземлювача η_n = 0,71. Дійсний опір протяжного заземлення становить:

$$R_n = \frac{R_{пр}}{\eta_n} = \frac{3,2}{0,71} = 4,5Ом \qquad (4.5)$$

Опір всього заземлюю чого пристрою становить:

$$R_u = \frac{R_c \cdot R_n}{R_c + R_n} = \frac{4.5 \cdot 7.3}{4.5 + 7.3} 2.78 < 3Ом \qquad (4.6)$$

Отже, число стержнів вибрано вірно.

4.4. Захист цивільного населення

Забезпечення захисту населення і території у разі загрози та виникнення надзвичайних ситуацій є одним з найважливіших завдань не лише підприємства, але й цілої держави. Актуальність проблеми забезпечення природо-техногенної безпеки населення і території зумовлена тенденціями зростання втрат людей і шкоди територіям, що спричиняються небезпечними природними явищами, промисловими аваріями і катастрофами.

Інженерний захист проводиться з метою виконання вимог ІТЗ із питань забудови міст, розміщення ПНО, будівлі будинків, інженерних споруд та інше.

Медичний захист проводиться для зменшення ступеня ураження людей, своєчасного надання допомоги постраждалим та їх лікування, забезпечення епідеміологічного благополуччя в районах надзвичайних ситуацій.

Біологічний захист включає своєчасне виявлення чинників біологічного зараження, їх характеру і масштабів, проведення комплексу адміністративно-господарських, режимно-обмежувальних і спеціальних протиепідемічних та медичних заходів.

Радіаційний і хімічний захист включає заходи щодо виявлення і оцінки радіаційної та хімічної обстановки, організацію і здійснення дозиметричного та хімічного контролю, розроблення типових режимів радіаційного захисту, забезпечення засобами індивідуального захисту, організацію і проведення спеціальної обробки.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Внаслідок роботи над даною кваліфікаційною роботою нами виконано всебічний аналіз розробки сучасного веб-додатку для управління персональними завданнями, підкреслюючи його актуальність у сучасному цифровому світі, що завдяки зростаючій потребі в ефективних інструментах управління часом через сплеск потоку інформації та щоденних обов'язків.

Технологічна основа пропонованої програми базується на сучасних стандартах веб-розробки, використовуючи React для інтерфейсу, щоб забезпечити адаптивний інтерфейс на основі компонентів, і Node.js із Firebase для серверної частини, щоб забезпечити масштабовану обробку даних у реальному часі та безпечну автентифікацію.

Вибір Firebase з його базою даних Firestore спрощує керування даними та синхронізацію, роблячи програму динамічною та зручною для користувача. Функціональні вимоги визначають такі функції, як створення завдань із такими параметрами, як пріоритет і крайні терміни, можливості пошуку та фільтрації, сповіщення про нагадування та аналітику діяльності для моніторингу продуктивності. Система також підтримує реєстрацію користувачів, синхронізацію даних між пристроями та планує майбутні вдосконалення, такі як групова співпраця та аналізи, керовані ШІ.

Процес розробки включав розробку модульного інтерфейсу зі зрозумілими сценаріями користувача, включаючи вхід, керування профілем, інтеграцію календаря та додавання завдань, оптимізоване для мобільного використання. Такі технології, як WebStorm IDE, Tailwind CSS і бібліотека FullCalendar, були використані для оптимізації розробки та забезпечення високоякісної взаємодії з користувачем. Інтерфейс було структуровано в окремі модулі, що полегшує навігацію та взаємодію. Загалом проект має на меті заповнити прогалини в існуючих рішеннях, пропонуючи просту, гнучку та комплексну платформу, адаптовану до індивідуальних потреб продуктивності, з акцентом на масштабованість, безпеку та зручність для користувачів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Стаття зі статистикою 2024 року щодо тайм-менеджменту [Електронний ресурс] https://development-academy.co.uk/news-tips/time-management- statistics-2024-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 впливит 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://collegeinfogeek.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/

16 Пістун І. П., Тимочко В.О., Городецький І. М.. Березовецький А. П. Охорона праці (гігієна праці та виробнича санітарія): навч. посібн. / за ред. І.П. Пістуна. Ч. І. Львів: Тріада плюс, 2011. 224 с.

17 Пістун І. П., Тимочко В.О., Городецький І. М.. Березовецький А. П. Охорона праці (гігієна праці та виробнича санітарія): навч. посібн. / за ред. І.П. Пістуна. Ч. ІІ. Львів: Тріада плюс, 2011. 274 с.

ДОДАДКИ

Коду веб-додатку для моніторингу особистих щоденних задач

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.firebaseapp.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;
```

AppRoter.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>

in`}>
up/step-1`}>

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

            <button className={styles['sign-in']}>Увійти</button>
        </Link>

        <Link classname={styles['button-link']} to={`/auth/sign-

            <button>Зареєструватись</button>
        </Link>

    </div>

</div>

<img
    className={styles['image-column']}
    src="./images/Schedule-cuate.svg" alt="icon"

            />
```

```
        </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)}> 'not-'}checked-circle.svg`}


${task.isDone ? styles['done'] : ''}`}>

<img src={`./images/${task.isDone ? '' :

    alt="circle"/>
<p className={`${styles['text']}

    {`${task.isHot ? '  ' : ''}

 ${task.name}: ${task.description ? task.description : '-'}`}
                                        </p>
```

```
                                        </div>

:)</span>}

    )}
</div>
: <span>На сьогодні нема ніяких планів. Відпочинь

                            <div className={styles['progress']}>
                                <CircularProgressbar
                                    value={percentage}
                                    text={`${percentage}%
                                    `}
                                    styles={buildStyles({
                                        textColor: "black",
                                        pathColor:
                                        "#677767",
                                        trailColor:
                                        "#F7F6F1"

                                                                    />
                                                                </div>

            </div>

                                        </div>

    </div>

</div>

})}


     );
 }

 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 категорії</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']}>

category.color,

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

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

alt="edit"/>

}}>

}

<img src="./images/edit-beg.svg"

                                        </Link>
```

```
                                        <img src="./images/delete-
                                             beg.svg" onClick={() =>
 deleteCategory(category.id, category.name)} alt="delete"/>
                                    </div>
                                </div>

                            )}

                        </div>

wrapper']}>

50}}

        <div onClick={toAddPage} className={styles['add-button-

            <img src="./images/add-beg.svg" alt="add"/>
        </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:

                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'
```

```
                                                                    />
                                                                </div>
        </div>
    </div>

</div>

}}
pointSize={5}
pointColor={{theme: 'background'}} pointBorderWidth={2} pointBorderColor={{from:
'serieColor'}} pointLabelYOffset={-12} areaBlendMode="screen"
areaBaselineValue={40}
areaOpacity={0}
useMesh={true}

         </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`}>
```

```
                    <img src="./images/user-beg.svg" alt="menu-icon"/>
                </Link>

                <Link to={`/home`}>
                    <img src="./images/home-beg.svg" alt="menu-icon"/>
                </Link>

                <Link to={`/calendar`}>
                    <img src="./images/calendar-beg.svg" alt="menu-icon"/>
                </Link>

                <Link to={`/add`}>
                    <img src="./images/add-beg.svg" alt="menu-icon"/>
                </Link>

icon"/>

<img src="./images/logout.svg" onClick={logout} alt="logout-

            </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"
                            />

    }

</div>

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

          </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: 'Всі події'},
            {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'
                                                                                    />
                                                                                </div>
</div>

}}
plugins={[dayGridPlugin]}
);
}
```