НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»
спеціальності 121 Інженерія програмного забезпечення
на тему: «Вебзастосунок «Schedule manager»»

Виконав: студент IV курсу, групи КП-02
Шилов Едуард Юрійович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Новак Дмитро Сергійович _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри
інформаційних технологій, штучного
інтелекту та кібербезпеки Національного університету
харчових технологій, к. т. н., доцент
Мошенський Андрій Олександрович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2024 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

«ЗАТВЕРДЖУЮ»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту
Шилову Едуарду Юрійовичу

1. Тема проєкту «Вебзастосунок «Schedule manager»», керівник проєкту Новак Дмитро Сергійович, к.т.н., доцент, затверджені наказом по університету від «30»травня 2024р. №2205-с
2. Термін подання студентом проєкту: «13» червня 2024 р.
3. Вихідні дані до проєкту**:** див. Технічне завдання.
4. Перелік задач, які мають бути вирішені**:**
   - провести аналіз вимог та проєктування системи;
   - розробити функціонал для керування розкладом;
   - тестування та налагодження;
5. Перелік обов’язкового ілюстративного матеріалу**:**
   - діаграма прецедентів (Use Case Diagram);
   - діаграма послідовності (Sequence Diagram);
   - діаграма класів (Class Diagram);
   - зразки коду та знімки екрану (Screenshots).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «30» жовтня 2023р

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи та питань, які мають бути розроблені відповідно до завдання	Термін виконання	Примітка
1.	Вивчення літератури за тематикою проєкту	05.04.2024	
2.	Розроблення та узгодження технічного завдання	19.04.2024	
3.	Розроблення структури програмного забезпечення	03.05.2024	
4.	Підготовка матеріалів першого розділу дипломного проєкту	10.05.2024	
5.	Розроблення дизайну системи та графічних елементів	20.05.2024	
6.	Підготовка матеріалів другого розділу дипломного проєкту	25.05.2024	
7.	Програмна реалізація дипломного проєкту	29.05.2024	
8.	Тестування програмного забезпечення	30.05.2024	
9.	Підготовка матеріалів третього розділу дипломного проєкту	01.06.2024	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	03.06.2024	
11.	Оформлення документації	05.06.2024	

Студент Едуард ШИЛОВ

Керівник проєкту  Дмитро НОВАК

АНОТАЦІЯ

У сучасному світі, коли життя стає дедалі більш динамічним та насиченим, ефективне управління часом набуває критичного значення. Саме тому був розроблений новітній вебзастосунок – "Schedule manager". Це потужний інструмент для планування, організації та відстеження виконання різноманітних завдань і подій. Його інтуїтивний календарний інтерфейс дозволяє користувачам у зручний спосіб створювати та впорядковувати власний розклад.

Одна з ключових особливостей додатку – широкий спектр інноваційних функцій. Користувачі мають змогу присвоювати пріоритети завданням, налаштовувати нагадування, класифікувати події за категоріями та застосовувати різноманітні фільтри для зручного перегляду розкладу. Крім того, розклад можна переглядати в денному, тижневому чи місячному режимах відображення.

Неабияку перевагу становить можливість двосторонньої синхронізації з популярними календарними сервісами, такими як Google Календар та Microsoft Exchange. Це забезпечує зібрання актуальних даних в єдиному місці, підвищуючи зручність планування та уникнення плутанини.

Маючи адаптивний та інтуїтивно зрозумілий користувацький інтерфейс Вебзастосунок забезпечує комфортну роботу як на десктопних пристроях, так і на мобільних гаджетах.

"Schedule manager" – це незамінний помічник у справі організації часу та підвищення продуктивності як у професійній сфері, так і в особистому житті. Завдяки практичному та зручному інтерфейсу, цей додаток стане корисним інструментом для кожного, хто прагне максимально ефективно виконувати свої завдання.

ABSTRACT

In the modern world, as life becomes increasingly dynamic and hectic, effective time management acquires critical importance. This is precisely why the cutting – edge web application "Schedule manager" was developed – a powerful tool for planning, organizing, and tracking the execution of various tasks and events. Its intuitive calendar interface allows users to conveniently create and organize their schedules.

One of the key features of the app is its wide range of innovative functions. Users have the ability to assign priorities to tasks, set up reminders, categorize events, and apply various filters for convenient schedule viewing. Additionally, the schedule can be viewed in daily, weekly, or monthly display modes.

A significant advantage is the capability for two-way synchronization with popular calendar services such as Google Calendar and Microsoft Exchange. This ensures the consolidation of up-to-date data in a single place, enhancing planning convenience and avoiding confusion.

With an adaptive and intuitively understandable user interface, the web application ensures comfortable operation on both desktop devices and mobile gadgets.

"Schedule manager" is an indispensable assistant in the matter of time organization and productivity enhancement, both in the professional sphere and in personal life. Thanks to its rich functionality and user-friendly interface, this app will become a useful tool for anyone who strives to execute their tasks as effectively as possible.

ДП.045490-01-90 Вебзастосунок “Schedule Manager”. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045490-02-91	Вебзастосунок “Schedule Manager”.	4	
	Технічне завдання.		
ДП.045490-03-81	Вебзастосунок “Schedule Manager”.	58	
	Пояснювальна записка		
ДП.045490-04-51	Вебзастосунок “Schedule Manager”.	4	
	Програма та методика тестування		
ДП.045490-05-34	Вебзастосунок “Schedule Manager”.	6	
	Керівництво користувача		
ДП.045490-06-99	Вебзастосунок “Schedule Manager”.	1	
	Діаграма розгортання.		
	Блок–схема алгоритму		
ДП.045490-07-99	Вебзастосунок “Schedule Manager”.		
	Інтерфейс користувача.	1	
	Структура взаємодії представлень		
ДП.045490-08-99	Вебзастосунок “Schedule Manager”.	1	
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

«____» _____ 2023 р.

ВЕБЗАСТОСУНОК "SHEDULE MANAGER"

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проєкту:

 Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Едуард ШИЛОВ

2023

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Вебзастосунок “Schedule Manager”.

Галузь застосування: Інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп’ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. МЕТА РОЗРОБКИ

Метою розробки є створення зручного веб–інструменту для ефективного планування, організації та моніторингу виконання різноманітних завдань і подій в єдиному календарному інтерфейсі.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Вебдодаток "Schedule Manager" має забезпечувати такі ключові функції:

1. Створення/редагування/видалення подій та завдань з пріоритетами та термінами.
2. Підтримка різних типів подій (одноразові, циклічні та інші).
3. Категоризація та фільтрація завдань.
4. Перегляд розкладу в різних режимах (день, тиждень, місяць).
5. Синхронізація з Google Calendar, Exchange тощо.
6. Імпорт/експорт даних розкладу.
7. Керування обліковими записами та розмежування доступу.

8. Адаптивний та зручний користувацький інтерфейс (UI/UX).
9. Забезпечення безпеки та надійності даних.
10. Додатково планується реалізувати такі функції:
    - Анімовані елементи інтерфейсу для підвищення зручності взаємодії.
    - Система сповіщень для нагадувань про наближення термінів.
    - Адаптивний дизайн інтерфейсу згідно з вимогами Google Material Design.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту має бути розроблена наступна документація:

1. Пояснювальна записка, яка міститиме опис предметної області, огляд існуючих рішень, обґрунтування вибраних технологій, опис архітектури додатку, особливостей реалізації, інструкції з розгортання та налаштування системи.
2. Програма та методика тестування вебдодатку, що включатиме тестові сценарії, стратегії тестування, вимоги до тестового середовища та звіти про проведене тестування.
3. Керівництво користувача, в якому буде докладно описано функціональні можливості, інструкції з встановлення та використання "Schedule Manager", а також посібник з вирішення типових проблем.

Графічна частина, що міститиме:

- Діаграму послідовності процесу планування завдань
- Діаграму компонентів архітектури додатку
- Макети користувацького інтерфейсу для ключових сторінок

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури та аналіз предметної області..... 05.04.2024
Розроблення та узгодження технічного завдання 19.04.2024
Проєктування архітектури вебдодатку та бази даних..... 29.04.2024
Розробка дизайну користувацького інтерфейсу та прототипів . 13.05.2024
Програмна реалізація клієнтської та серверної частин додатку 17.05.2024
Тестування функціональності "Schedule Manager” 26.05.2024
Підготовка матеріалів пояснювальної записки 03.06.2024
Підготовка графічної частини та діаграм проєкту 05.06.2024
Оформлення проєктної документації згідно з вимогами..... 10.06.2024

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2024 р.

ВЕБЗАСТОСУНОК "SCHEDULE MANAGER"

Пояснювальна записка

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проєкту:

 Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Едуард ШИЛОВ

2024

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

API – набір визначень, протоколів та засобів для взаємодії з певним програмним забезпеченням.

CRUD – базові операції (Create, Read, Update, Delete) для роботи з даними в інформаційних системах.

DBMS (Database Management System) – система керування базами даних.

HTTP – протокол передачі даних для веб–ресурсів.

JSON – текстовий формат обміну даними, заснований на JavaScript.

MVP (Model-View-Presenter) – архітектурний шаблон для розробки інтерфейсів користувача.

REST (Representational State Transfer) – архітектурний стиль для розробки веб-сервісів.

SPA (Single Page Application) – вебдодаток, що завантажується на одній сторінці та динамічно оновлює контент без перезавантаження.

UI (User Interface) – користувацький інтерфейс, засіб взаємодії між людиною та програмним забезпеченням.

UX (User Experience) – досвід взаємодії користувача з продуктом, що включає зручність, ефективність та задоволеність.

Адаптивний дизайн – підхід до веб–дизайну, який забезпечує оптимальне відображення та функціонування на різних пристроях та розмірах екрану.

Анімація – імітація руху за допомогою послідовності статичних зображень.

Календарний сервіс – онлайн–сервіс для планування та відстеження подій у календарі.

Категоризація – групування об'єктів за певними ознаками або категоріями.

ВСТУП

Ефективне управління часом та підвищення продуктивності є надзвичайно важливими аспектами в сучасному світі. Наше життя наповнене безліччю завдань, проєктів та подій, які потребують ретельного планування та організації робочого процесу. Відсутність контролю над розкладом може спричинити хаос, стрес та зниження продуктивності як на роботі, так і в особистому житті.

Протягом останніх років з'явилася велика кількість додатків та онлайн-сервісів для планування завдань. Однак більшість з них мають обмежений функціонал або незручний інтерфейс. Деякі пропонують лише базові можливості створення та відстеження завдань, тоді як інші переобтяжені зайвими функціями, що ускладнює їх використання.

Дипломний проєкт "Schedule Manager" покликаний створити зручний і потужний веб–інструмент для ефективного управління часом та організації робочого процесу. Цей додаток поєднує в собі інтуїтивний користувацький інтерфейс та сучасні веб-технології.

"Schedule Manager" дозволяє створювати, планувати та відстежувати виконання різноманітних завдань в єдиному календарному інтерфейсі. Він пропонує широкий спектр функцій, таких як встановлення пріоритетів, категоризація завдань, налаштування нагадувань та фільтрація подій. Додаток також підтримує різні режими перегляду розкладу.

Важливою перевагою є інтеграція з популярними календарними сервісами, що дозволяє зберігати актуальні дані про події в одному місці, уникаючи плутанини.

Інтуїтивний та адаптивний інтерфейс гарантує зручність роботи на різних пристроях, відповідаючи сучасним тенденціям веб-дизайну та принципам юзабіліті.

Цей дипломний проєкт є спробою вирішити проблему ефективного розподілу часу та ресурсів за допомогою новітніх веб-технологій та інноваційних підходів до планування завдань.

1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Актуальність задачі ефективного управління часом

Ефективне управління часом є комплексом методів, стратегій і технік, спрямованих на оптимізацію використання часу для досягнення конкретних цілей і задач. Це включає в себе процес планування, організації, мотивації та контролю часу, з метою підвищення продуктивності та ефективності як у професійній діяльності, так і в особистому житті. Ефективне управління часом дозволяє індивідам та організаціям максимізувати використання наявних ресурсів, мінімізувати витрати часу на непродуктивні діяльності та забезпечити своєчасне виконання завдань. До основних елементів управління часом належать визначення пріоритетів, встановлення цілей, розробка планів дій, делегування завдань та контроль за їх виконанням.

Управління часом набуло особливої актуальності в сучасному світі, що характеризується швидким темпом життя, значним обсягом інформації та високими вимогами до продуктивності. Уміння ефективно управляти часом стає ключовим фактором успіху в різних сферах діяльності.

По–перше, управління часом є необхідним інструментом для підвищення продуктивності та ефективності. В умовах зростаючої конкуренції на ринку праці та постійних змін у бізнес-середовищі, здатність оптимально розподіляти свій час дозволяє індивідам і організаціям виконувати більше завдань за коротший проміжок часу, що, в свою чергу, підвищує їхню конкурентоспроможність.

По-друге, ефективне управління часом сприяє зменшенню рівня стресу та покращенню психічного здоров’я. Чітке планування та організація діяльності дозволяють уникати перевантаження, запобігають накопиченню незавершених завдань та створюють відчуття контролю над власним життям. Це особливо важливо в умовах сучасного суспільства, де високий рівень стресу є однією з головних причин зниження якості життя.

По-третє, управління часом допомагає досягти балансу між професійним та особистим життям. Ефективне розподілення часу дозволяє знайти час для відпочинку, сім'ї, хобі та інших важливих аспектів життя, що сприяє загальному задоволенню та гармонії.

Високий темп розвитку технологій також вносить свої корективи в управління часом. Нові інструменти та програми для планування, моніторингу та аналізу часу дозволяють більш ефективно керувати своїм часом та адаптуватися до швидкоплинних змін у професійному середовищі.

В останні десятиліття світ зіткнувся зі значним зростанням обсягу інформації, яку потрібно обробляти та аналізувати. В умовах інформаційного вибуху, де кількість доступних даних постійно зростає, управління часом стає критично важливим для ефективного функціонування. Постійне надходження нової інформації вимагає від працівників швидкої адаптації та обробки, що збільшує навантаження на їхні когнітивні ресурси. Здатність структурувати свій час дозволяє фахівцям уникати перевантаження інформацією, зосереджуватися на пріоритетних завданнях та підвищувати продуктивність. Таким чином, управління часом сприяє ефективнішому використанню ресурсів і досягненню кращих результатів у професійній діяльності.

Сучасний світ праці зазнав значних змін, що також підвищує актуальність управління часом. Поширення віддаленої роботи та гнучких графіків вимагає від працівників нових навичок організації власного часу. Віддалена робота, з одного боку, надає більше свободи в плануванні робочого дня, але з іншого – потребує високого рівня самодисципліни та вміння встановлювати межі між роботою та особистим життям. Гнучкі графіки дозволяють адаптувати робочий час до індивідуальних потреб, але вимагають ефективного планування для уникнення хаосу та забезпечення виконання завдань у визначені строки. В таких умовах управління часом

стає необхідним інструментом для підтримання балансу та забезпечення продуктивності.

Психологічні аспекти також відіграють важливу роль в актуальності управління часом. Нездатність ефективно керувати своїм часом часто призводить до хронічного стресу, що негативно впливає на психічне та фізичне здоров'я. Постійне відчуття браку часу, тиск невиконаних завдань та перевантаження можуть стати причиною професійного вигорання, тривожних розладів та інших психосоматичних проблем. Управління часом дозволяє зменшити рівень стресу, забезпечуючи чітку структуру дня, що допомагає уникнути перенавантаження та створює відчуття контролю над ситуацією. Це особливо важливо в умовах сучасного світу, де високий темп життя та постійні зміни є постійними джерелами стресу.

Таким чином, зростання обсягу інформації та робочих завдань, зміни в умовах праці та життя, а також психологічні аспекти та стрес є ключовими факторами, що підвищують актуальність управління часом у сучасному світі. Здатність ефективно керувати своїм часом дозволяє зберігати продуктивність, підтримувати баланс між роботою та особистим життям, а також зменшувати рівень стресу, сприяючи загальному покращенню якості життя.

Таким чином, управління часом є критично важливим для досягнення успіху та гармонії в сучасному світі, забезпечуючи баланс між різними аспектами життя та сприяючи підвищенню загальної якості життя.

1.2. Огляд існуючих додатків для планування завдань

Todoist – пропонує можливість інтеграції з багатьма іншими сервісами, такими як Google Calendar і Slack, для підвищення продуктивності. Додаток підтримує роботу в режимі офлайн, дозволяючи користувачам керувати своїми завданнями навіть без підключення до Інтернету. Зручний інтерфейс і регулярні оновлення роблять Todoist відмінним вибором для індивідуального та командного використання.

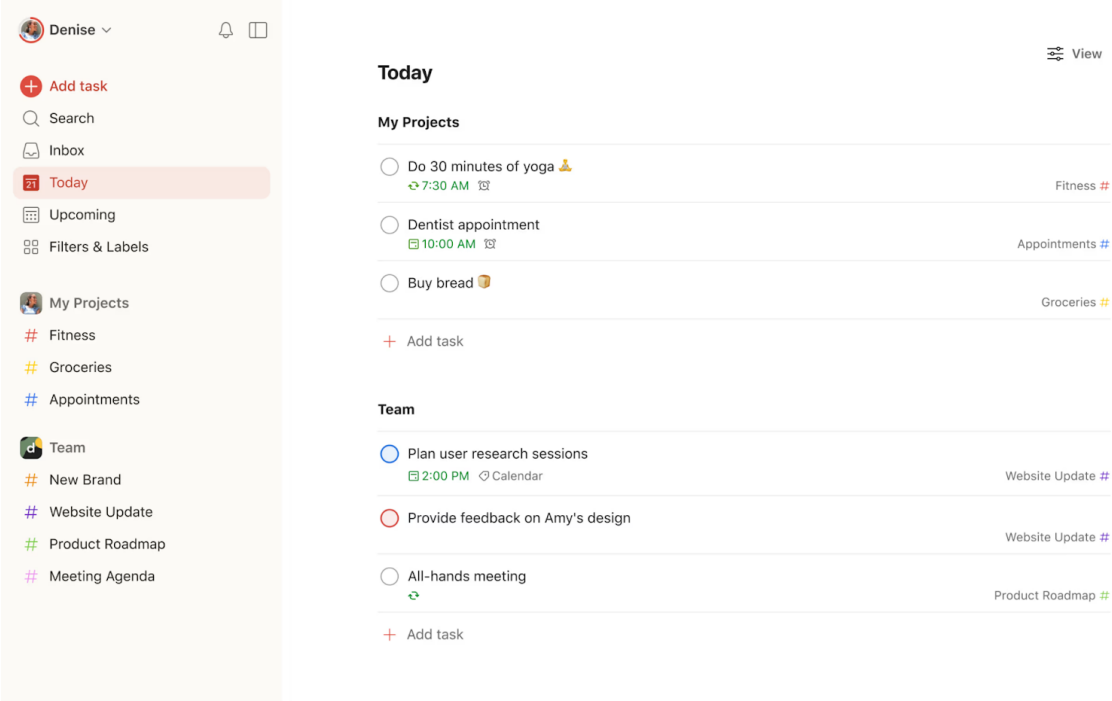


Рис. 1.1. Todoist

Переваги:

- Інтуїтивно зрозумілий інтерфейс.
- Підтримка кросплатформенності (доступний на Windows, macOS, iOS, Android).
- Можливість спільної роботи над завданнями.
- Інтеграція з іншими сервісами (Google Calendar, Slack, etc.).

Недоліки:

- Обмежений функціонал в безкоштовній версії.
- Деякі користувачі відзначають, що інтерфейс може бути перевантаженим при великій кількості завдань.

Trello – дозволяє командам створювати дошки для різних проєктів і використовувати картки для завдань, що спрощує відстеження прогресу. Інтеграції з іншими інструментами, такими як Slack і Google Drive, допомагають централізувати робочий процес. Завдяки своїй гнучкості та простоті використання, Trello підходить для як для особистого, так і для командного управління проєктами.

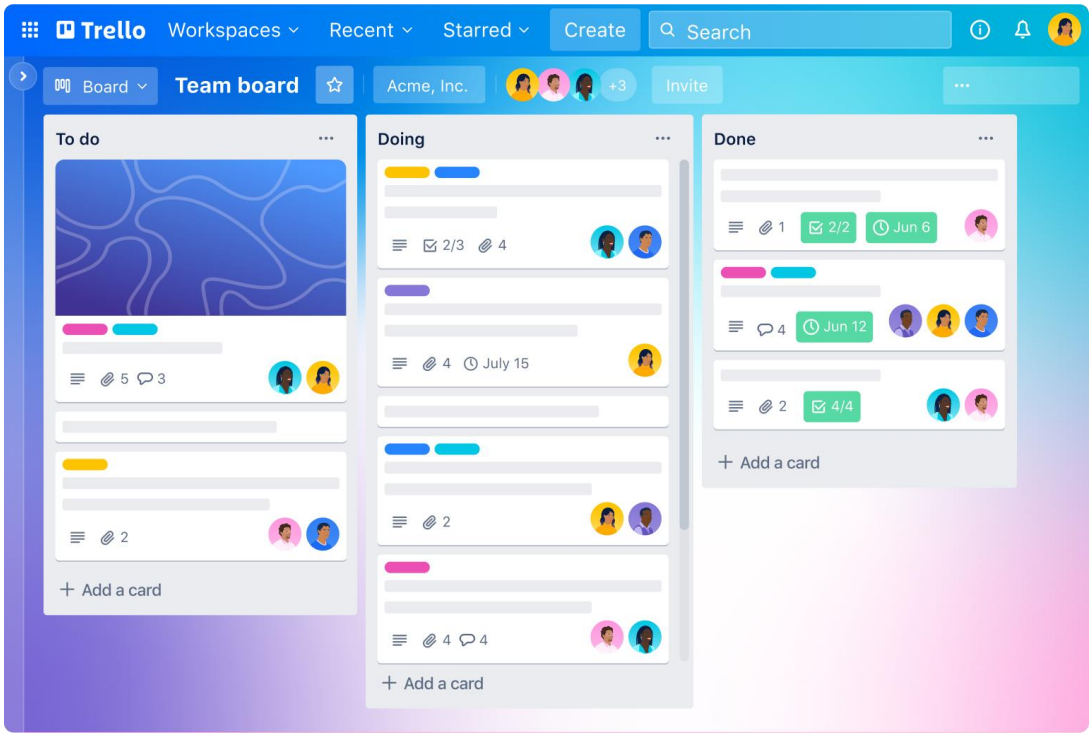


Рис. 1.2. Trello

Переваги:

- Візуальна та інтуїтивно зрозуміла система організації завдань.
- Підтримка командної роботи.
- Безкоштовна версія з широким функціоналом.
- Інтеграція з багатьма сторонніми сервісами (Google Drive, Jira, etc.).

Недоліки:

- Може бути менш зручним для управління великою кількістю дрібних завдань.
- Обмежені можливості аналітики та звітності.

Microsoft To Do – це простий і зручний додаток для управління завданнями від компанії Microsoft, інтегрований з іншими продуктами Microsoft. Він дозволяє створювати списки завдань, встановлювати нагадування та пріоритети для ефективнішого планування часу. Завдяки інтеграції з Outlook та іншими сервісами Microsoft, користувачі можуть легко синхронізувати свої завдання між різними пристроями.

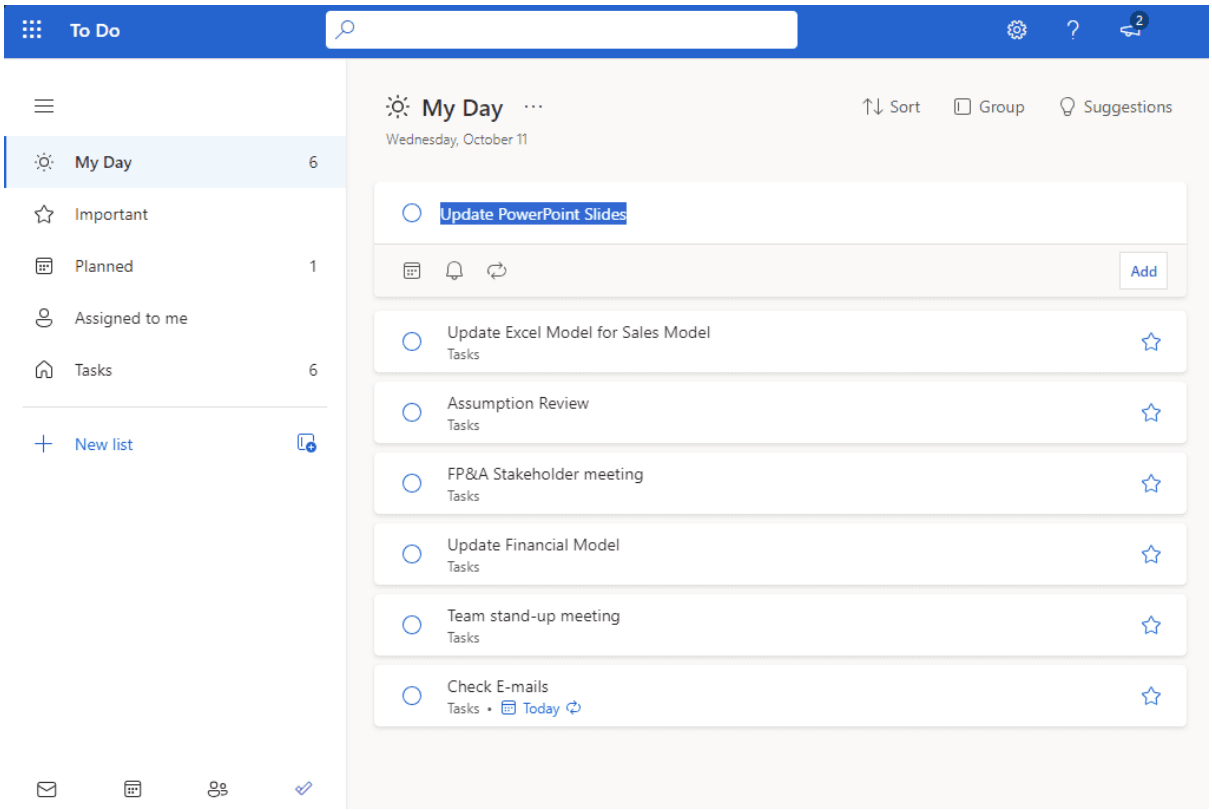


Рис. 1.3. Microsoft To Do

Переваги:

- Глибока інтеграція з екосистемою Microsoft (Outlook, OneNote).
- Простий та чистий інтерфейс.
- Безкоштовний.

Недоліки:

- Обмежені можливості налаштування.
- Відсутність деяких просунутих функцій, таких як нагадування на основі місця.

Asana – це потужний інструмент для управління проєктами, який дозволяє організовувати роботу в команді, відслідковувати прогрес та керувати завданнями. Також пропонує мобільний додаток, що дозволяє керувати проєктами на ходу. Система сповіщень допомагає залишатися в курсі всіх оновлень і змін у проєктах.

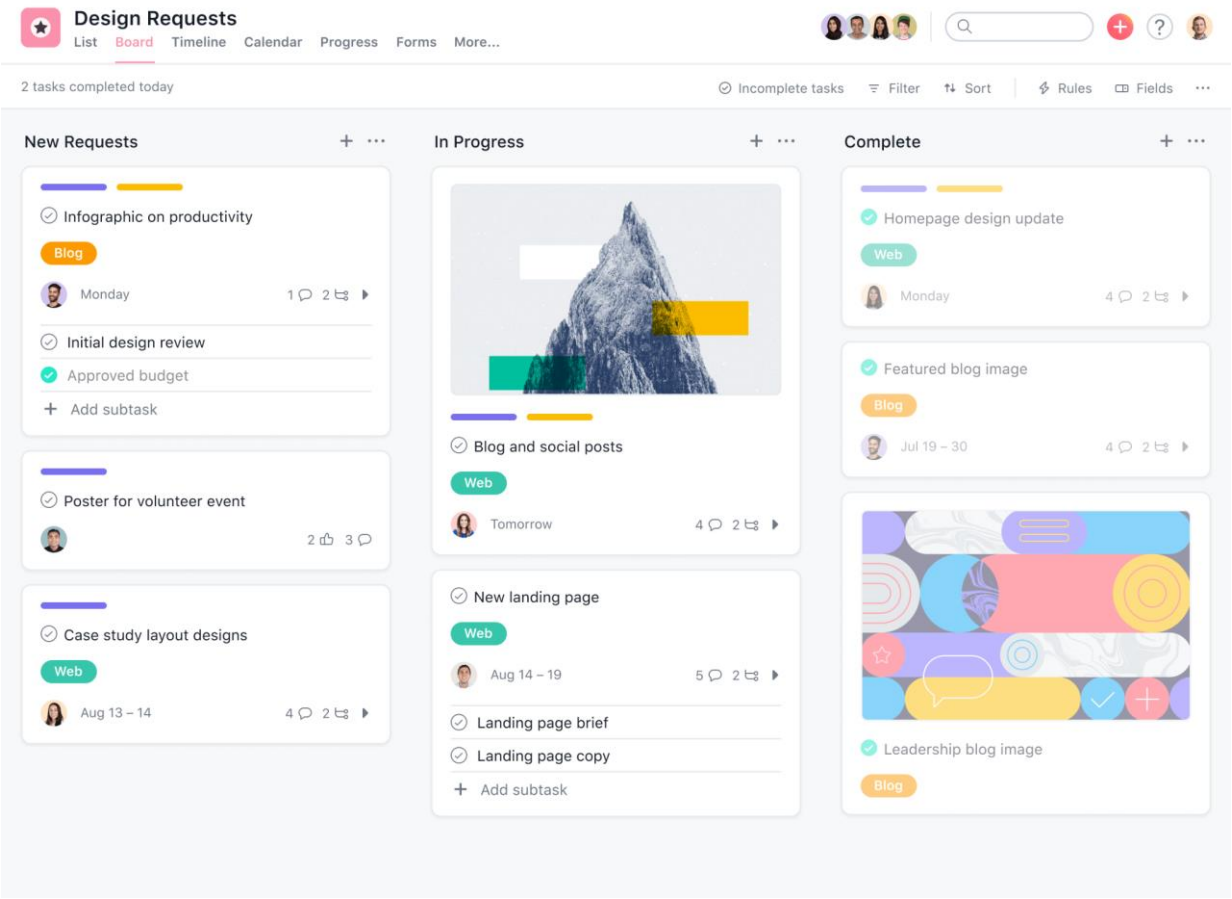


Рис. 1.4. Asana

Переваги:

- Потужний набір інструментів для управління проєктами.
- Підтримка командної роботи та співпраці.
- Інтеграція з багатьма сервісами (Slack, Google Drive, etc.).

Недоліки:

- Може бути занадто складним для невеликих проєктів.
- Деякі функції доступні лише в платній версії.

Notion – це універсальний інструмент для організації нотаток, завдань і проєктів, що поєднує в собі функції бази даних, календаря і редактора документів. Notion дозволяє створювати індивідуальні робочі простори, які можуть адаптуватися до будь-яких робочих процесів і потреб, забезпечуючи максимальну ефективність і організацію.

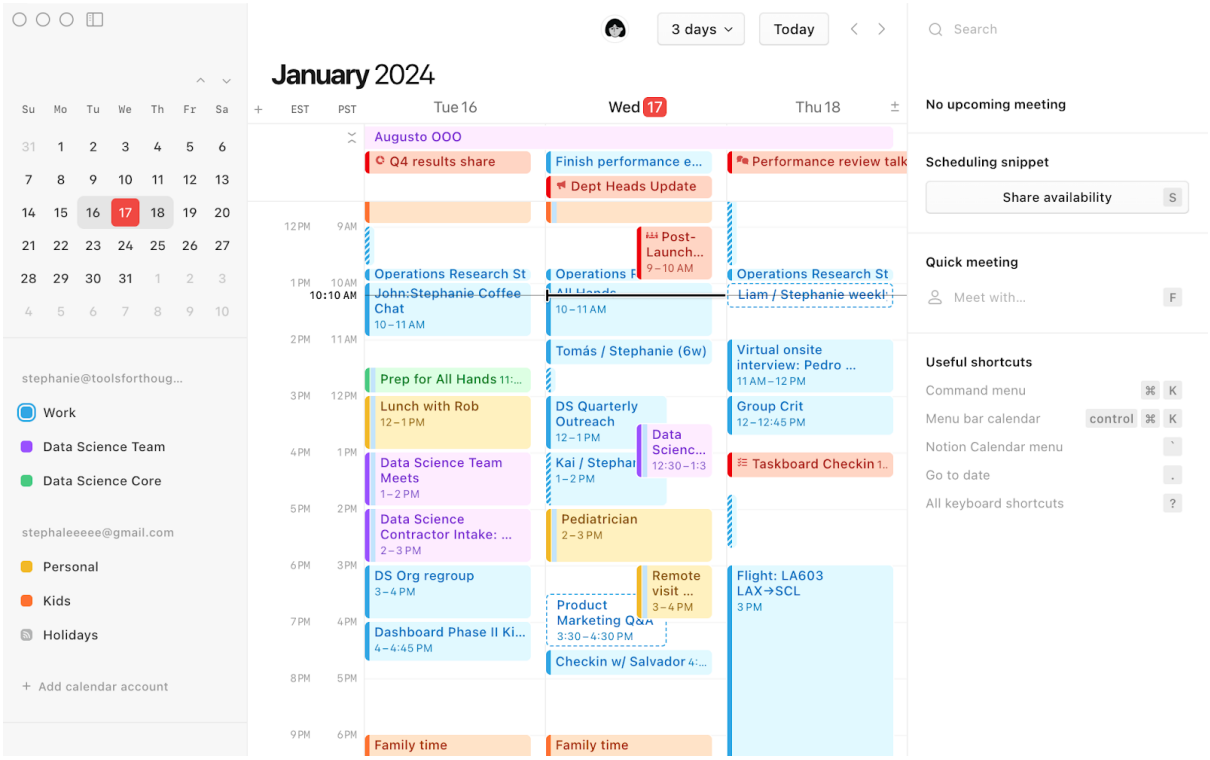


Рис. 1.5. Notion

Переваги:

- Високий рівень налаштування та гнучкості.
- Об'єднання функцій багатьох додатків в одному інструменті.
- Можливість співпраці в реальному часі.

Недоліки:

- Може мати круту криву навчання.
- Деякі користувачі відзначають, що додаток може працювати повільно при великій кількості даних.

1.3. Обґрунтування необхідності створення "Schedule Manager"

Існуючі додатки для планування завдань, хоча і мають широкий функціонал, не завжди можуть задовольнити унікальні потреби конкретної аудиторії. Створення власного "Schedule Manager" дозволить врахувати специфічні вимоги користувачів, такі як індустріальні особливості, корпоративні стандарти або індивідуальні уподобання.

У додатку можна впровадити більш суворі заходи безпеки для захисту конфіденційної інформації, ніж це пропонують стандартні рішення. Це особливо важливо для компаній, які працюють з чутливими даними і мають високі вимоги до інформаційної безпеки.

Розробка власного "Shedule Manager" забезпечує повну гнучкість у додаванні нових функцій та масштабуванні системи відповідно до зростаючих потреб бізнесу. Стандартні додатки можуть мати обмеження в розширенні функціоналу або бути недостатньо гнучкими для адаптації до нових викликів.

Власний додаток дозволяє створити інтерфейс, максимально адаптований до потреб користувачів. Це може включати спрощений процес введення даних, персоналізовані налаштування, спеціалізовані функції для конкретних завдань, що значно підвищить ефективність роботи користувачів.

Створення власного "Schedule Manager" є стратегічно виправданим рішенням, яке дозволить врахувати унікальні потреби користувачів, забезпечити високу безпеку даних, інтеграцію з внутрішніми системами, гнучкість та масштабованість, оптимізацію користувацького досвіду, конкурентні переваги та економічну доцільність.

2. ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

2.1. Вибір стеку технологій для веб-розробки

При розробці простої веб-програми можна скористатися основними технологіями, що забезпечують швидкий розвиток, простоту використання та хорошу інтеграцію. Вибраний стек технологій включає в себе:

- Фронтенд: HTML, CSS, JavaScript.
- Бекенд: Firebase.
- Аутентифікація: Firebase Authentication.
- База даних: Firestore.

Вибраний стек технологій для веб-розробки включає HTML, CSS та JavaScript для клієнтської частини, Firebase для бекенду, Firebase Authentication для аутентифікації користувачів та Firestore для зберігання даних.

HTML є основною мовою розмітки для створення структури веб-сторінок і підтримується всіма браузерами. Це стандартна і проста у використанні технологія, яка дозволяє створювати структурований контент. CSS використовується для стилізації HTML-елементів, роблячи їх привабливими та адаптивними. Воно відокремлює структуру документа від його стилю, що полегшує підтримку та оновлення, а також забезпечує адаптивність на різних пристроях. JavaScript, мова програмування для створення динамічних і інтерактивних веб-сторінок, дозволяє додавати взаємодію та функціональність, працює у всіх сучасних браузерах і легко інтегрується з іншими веб-технологіями.

Firebase є хмарним рішенням, яке забезпечує безсерверну архітектуру, що зменшує необхідність у налаштуванні та управлінні серверами. Firebase підтримує роботу в реальному часі, що дозволяє створювати інтерактивні застосунки з миттєвим оновленням даних. Це рішення легко інтегрується з іншими сервісами Firebase, такими як

Firestore та Firebase Authentication, і забезпечує автоматичне масштабування в залежності від навантаження.

Firebase Authentication надає просте та швидке налаштування аутентифікації користувачів. Воно підтримує різні методи аутентифікації, включаючи логін через email/пароль, соціальні мережі (Google, Facebook тощо) та інші провайдери. Це рішення має вбудовані механізми для забезпечення безпеки даних користувачів і легко інтегрується з іншими сервісами Firebase та Firestore для зберігання даних користувачів.

Firestore є NoSQL базою даних, що забезпечує гнучку структуру зберігання даних у вигляді колекцій і документів, що особливо підходить для динамічних застосунків. Вона підтримує синхронізацію даних у реальному часі між клієнтом і сервером, автоматично масштабується відповідно до потреб застосунку та безшовно інтегрується з іншими сервісами Firebase, такими як Firebase Authentication та Firebase Hosting.

Таким чином, цей стек технологій забезпечує простоту, ефективність і швидкість розробки, дозволяючи створювати інтуїтивний і адаптивний інтерфейс користувача, забезпечуючи надійне і гнучке зберігання даних і підтримку в реальному часі, а також легке налаштування безпечної аутентифікації користувачів.

2.2. Вибір фреймворку для клієнтської частини

HTML є основною мовою розмітки для створення веб-сторінок. Вона використовується для структурування вмісту веб-сторінки за допомогою тегів.

Основні особливості:

1. Теги: Основні будівельні блоки HTML. Наприклад, <p> для абзацу, <a> для посилання, для контейнера і т.д.
2. Атрибути: Використовуються для додавання додаткової інформації до елементів HTML, наприклад, id, class, src, href.

3. Документ: Структура HTML–документа включає основні теги, такі як <!DOCTYPE html>, <html>, <head>, <title>, <body>.

CSS використовується для стилізації та оформлення даних HTML-документів. Вона визначає, як елементи повинні відображатися на екрані.

Основні особливості:

1. Селектори – вказують, які елементи HTML будуть стилізовані. Наприклад, селектор p вибирає всі абзаци.
2. Властивості та значення – визначають стиль елемента. Наприклад, color, font-size, margin, padding.
3. Каскадність – визначає, як застосовуються стилі, якщо вони конфліктують, з врахуванням специфічності, порядку, та спадковості.

JavaScript – це мова програмування, яка використовується для додавання інтерактивності та динамічної поведінки на веб-сторінках.

Основні особливості:

1. Можливість змінювати вміст HTML та стилі CSS під час виконання.
2. Можливість реагувати на дії користувача, такі як натискання кнопок, пересування миші, введення тексту.
3. Основні блоки коду, які можна викликати в будь–який момент.
4. JavaScript підтримує об'єктно–орієнтоване програмування.

Переваги:

- HTML – основна мова розмітки для створення структури веб-сторінок.
- CSS – використовується для стилізації HTML-елементів і забезпечення адаптивного дизайну.
- JavaScript – мова програмування для створення динамічних та інтерактивних елементів на веб-сторінках.

Простота використання:

- Легко навчитися і почати використовувати.
- Висока сумісність з усіма браузерами.
- Широка підтримка і багато доступних бібліотек.

Використання:

- Bootstrap – CSS-фреймворк для швидкого створення адаптивного дизайну.
- Vanilla JavaScript – використання чистого JavaScript для додавання інтерактивності без додаткових фреймворків.

2.3. Вибір технології для серверної частини

Firebase – це платформа для розробників мобільних та вебдодатків, створена компанією Google. Вона надає набір інструментів і сервісів, які полегшують створення, розгортання та управління додатками. Firebase дозволяє розробникам зосередитися на створенні функціональності додатків без необхідності турбуватися про інфраструктуру, сервери чи бази даних.

Переваги:

- Не потребує налаштування серверів, автоматичне масштабування.
- Підтримка роботи в реальному часі, що дозволяє створювати інтерактивні застосунки.
- Легко інтегрується з клієнтською частиною через SDK.

Використання:

- Надає можливість швидко створити бекенд для вебзастосунків.
- Містить вбудовані рішення для автентифікації, баз даних та хостингу.

2.4. Вибір системи керування базами даних

Firestore, також відомий як Cloud Firestore, є гнучкою, масштабованою базою даних для мобільних, веб- та серверних розробок, яка належить до сімейства Firebase. Firestore дозволяє зберігати, синхронізувати та запитувати дані для ваших додатків на глобальному рівні. Вона є покращеною версією Firebase Realtime Database з багатьма додатковими функціями та можливостями.

Переваги:

- NoSQL база даних – гнучка структура, що зберігає дані у вигляді колекцій і документів.
- Дані синхронізуються в реальному часі між клієнтом і сервером.
- Автоматичне масштабування в залежності від навантаження.

Використання:

- Використовується для зберігання даних про розклад, задачі, користувачів тощо.
- Легка інтеграція з іншими сервісами Firebase.

Firestore легко інтегрується з іншими сервісами Firebase, такими як Firebase Authentication, Firebase Functions, Firebase Storage, що дозволяє створювати комплексні та багатофункціональні додатки.

2.5. Вибір допоміжних бібліотек та інструментів

Bootstrap – це популярний фреймворк для створення адаптивних вебсайтів та вебдодатків. Він містить набір інструментів для роботи з HTML, CSS та JavaScript, що допомагає розробникам швидко створювати стильні та функціональні веб-сторінки з мінімальними зусиллями.

Bootstrap побудований з використанням системи сіток (grid system), яка дозволяє створювати макети, що автоматично адаптуються до різних розмірів екранів, від мобільних пристроїв до настільних комп'ютерів. Bootstrap включає в себе безліч готових компонентів, таких як навігаційні бари, каруселі, модальні вікна, спливаючі підказки (tooltips), випадаючі

меню (dropdowns) та багато інших. Ці компоненти допомагають швидко додавати складну функціональність до веб-сторінок. Фреймворк дозволяє легко налаштовувати зовнішній вигляд сайтів за допомогою тем та змінних. Можна змінювати кольори, шрифти та інші стилі відповідно до потреб проєкту.

Переваги:

- Набір готових компонентів для швидкого створення інтерфейсу.
- Забезпечує адаптивність на різних пристроях.
- Прості класи для стилізації елементів.

Firebase Authentication – це сервіс, що надається платформою Firebase для аутентифікації користувачів у веб- та мобільних додатках. Він підтримує різні методи аутентифікації, такі як електронна пошта і пароль, телефонні номери, а також соціальні логіни, такі як Google, Facebook, Twitter та інші. Цей сервіс спрощує процес реєстрації та входу користувачів, а також забезпечує високу безпеку та надійність.

Методи аутентифікації:

- Електронна пошта і пароль.
- Класичний метод аутентифікації, де користувач реєструється або входить за допомогою своєї електронної пошти та пароля.
- Телефонний номер.
- Аутентифікація за допомогою OTP (одноразового пароля), відправленого на телефон користувача.
- Соціальні логіни.
- Підтримка аутентифікації через популярні соціальні мережі та сервіси, такі як Google, Facebook, Twitter, GitHub, Microsoft, Apple та інші.
- Анонімна аутентифікація дозволяє користувачам взаємодіяти з додатком без обов'язкової реєстрації, що може бути зручним для тимчасових або тестових користувачів.

Переваги:

- Легко налаштувати автентифікацію користувачів через різні методи (email/пароль, Google, Facebook тощо).
- Вбудовані механізми безпеки для захисту даних користувачів.
- Легка інтеграція з іншими сервісами Firebase.

Firebase Hosting – це швидка та безпечна платформа для хостингу вебдодатків та статичних сайтів. Вона пропонує глобальну мережу доставки контенту (CDN), автоматичне використання HTTPS, безшовну інтеграцію з іншими сервісами Firebase та простий процес розгортання.

Переваги:

- Легко хостити статичні файли (HTML, CSS, JS) і забезпечити швидкий доступ до них.
- Вбудована підтримка SSL для захищеного з'єднання.
- Автоматичне масштабування в залежності від трафіку.

Firebase Hosting використовує глобальну мережу доставки контенту, що забезпечує високу швидкість завантаження та низьку затримку для користувачів з усього світу. Всі сайти, розміщені на Firebase Hosting, автоматично використовують HTTPS, що забезпечує безпечний зв'язок між користувачами та вашим сайтом. Легка інтеграція з Firebase Authentication, Firestore, Realtime Database, Cloud Functions та іншими сервісами для створення потужних вебдодатків.

3. АРХІТЕКТУРА ТА ДИЗАЙН ДОДАТКУ

3.1. Загальна архітектура вебдодатку

Вебдодатки сучасного типу потребують ретельно спланованої архітектури, яка забезпечить гнучкість, масштабованість, безпеку та інтерактивність. В даному розділі буде описано загальну архітектуру вебдодатку, створеного з використанням стеку технологій, вибраного у попередніх розділах. Архітектура розглядає такі компоненти: клієнтську частину, серверну частину, базу даних, аутентифікацію та хостинг.

Клієнтська частина

Клієнтська частина вебдодатку побудована за допомогою HTML, CSS та JavaScript. HTML (HyperText Markup Language) відповідає за структурування контенту на веб-сторінках. Основні теги HTML, такі як , <p>, <a>, використовуються для визначення різних елементів сторінки. CSS (Cascading Style Sheets) відповідає за стилізацію HTML-документів, роблячи їх візуально привабливими та адаптивними для різних пристроїв. Використовуючи селектори, властивості та значення CSS, можна легко змінювати зовнішній вигляд елементів HTML. JavaScript додає інтерактивність до веб-сторінок, дозволяючи змінювати вміст та стилі елементів HTML під час виконання, а також реагувати на дії користувачів.

Для покращення продуктивності та зручності розробки, до клієнтської частини додано фреймворк Bootstrap. Bootstrap, популярний CSS-фреймворк, дозволяє швидко створювати адаптивні дизайни за допомогою системи сіток (grid system) та готових компонентів. Це значно скорочує час розробки та забезпечує узгоджений вигляд інтерфейсу на різних пристроях.

Серверна частина

Серверна частина вебдодатку реалізована на платформі Firebase. Firebase, розроблена компанією Google, надає набір інструментів та сервісів, які спрощують створення, розгортання та управління

вебдодатками. Однією з ключових переваг Firebase є безсерверна архітектура, що зменшує потребу в налаштуванні та управлінні фізичними серверами. Це дозволяє зосередитися на функціональності додатку, а не на інфраструктурних питаннях.

Firebase Hosting використовується для розгортання та хостингу вебдодатку. Цей сервіс забезпечує швидке та безпечне доставлення контенту через глобальну мережу доставки контенту (CDN), автоматичне використання HTTPS та легке масштабування в залежності від трафіку. Це забезпечує високий рівень доступності та безпеки вебдодатку.

База даних

Для зберігання даних використовується Firestore, яка є NoSQL базою даних. Firestore забезпечує гнучку структуру зберігання даних у вигляді колекцій та документів. Така структура особливо підходить для динамічних застосунків, де дані можуть швидко змінюватися та синхронізуватися в реальному часі між клієнтом та сервером. Firestore автоматично масштабується відповідно до навантаження, що забезпечує стабільну роботу додатку навіть при значних обсягах даних та високому трафіку.

Аутентифікація

Аутентифікація користувачів здійснюється за допомогою Firebase Authentication. Цей сервіс підтримує різні методи аутентифікації, включаючи логін через email/пароль, телефонний номер та соціальні мережі (Google, Facebook тощо). Firebase Authentication забезпечує високий рівень безпеки та легко інтегрується з іншими сервісами Firebase, що дозволяє зберігати дані користувачів у Firestore та використовувати їх для управління доступом до різних функцій додатку.

Інтеграція компонентів

Основна інтеграція між компонентами вебдодатку відбувається через Firebase SDK (Software Development Kit), який забезпечує зручний доступ до всіх сервісів Firebase, включаючи Authentication, Firestore та Hosting.

Використовуючи Firebase SDK, розробники можуть легко реалізувати функції аутентифікації, зберігання та синхронізації даних без необхідності писати складний бекенд-код. Для ілюстрації архітектури вебдодатку нижче представлено діаграму розгортання, створену за допомогою мови UML.

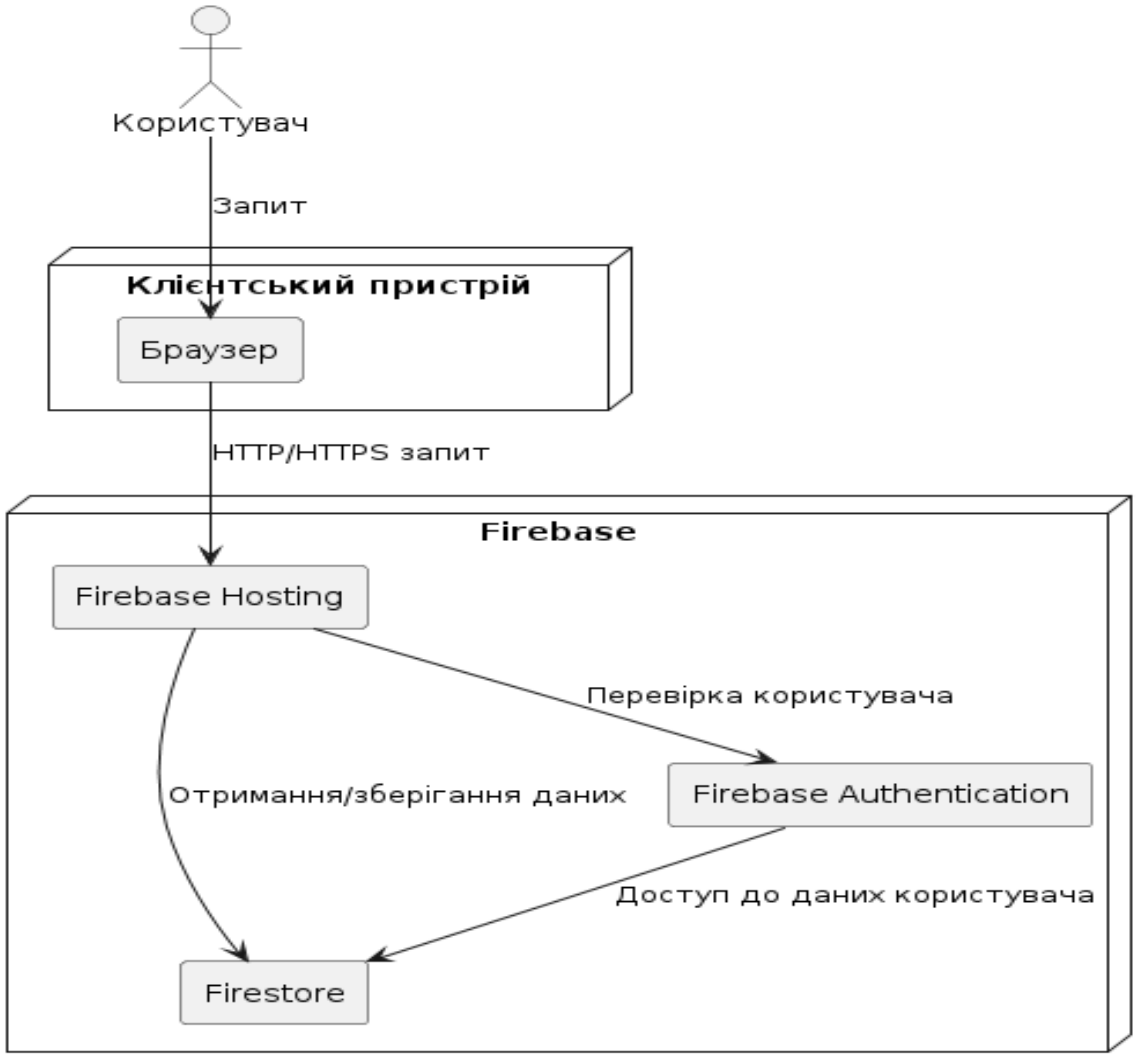


Рис. 3.1. Діаграма розгортання

Діаграма компонентів демонструє взаємодію різних частин вебдодатку. Вона візуалізує, як окремі модулі, такі як інтерфейс користувача, серверна логіка і база даних, взаємодіють один з одним. Така діаграма допомагає розробникам і архітекторам систем краще зрозуміти структуру додатку та залежності між компонентами. Вона також сприяє виявленню можливих точок відмови і зон для оптимізації. Використання

діаграми компонентів полегшує планування, реалізацію та підтримку складних систем.

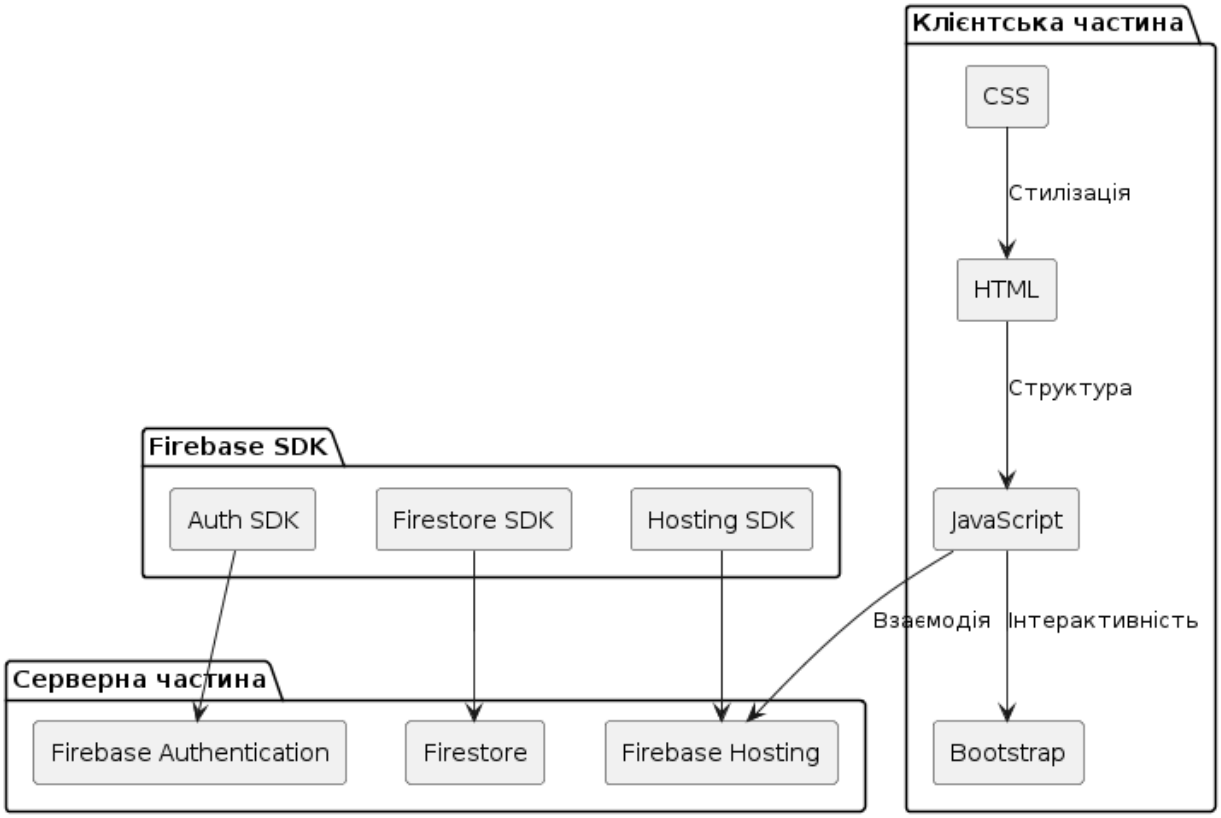


Рис. 3.2. Діаграма компонентів

Запропонована архітектура вебдодатку базується на сучасних технологіях та практиках, які забезпечують високу продуктивність, безпеку та масштабованість. Використання Firebase для серверної частини дозволяє зменшити час на налаштування інфраструктури та зосередитися на розробці функціональності.

Комбінація HTML, CSS, JavaScript та Bootstrap для клієнтської частини забезпечує створення інтуїтивно зрозумілого та адаптивного інтерфейсу користувача. Firestore, яка є частиною Firebase, надає гнучке та надійне зберігання даних з можливістю синхронізації в реальному часі. Firebase Authentication спрощує процес аутентифікації користувачів та забезпечує високу безпеку. Така архітектура дозволяє створювати сучасні вебдодатки, які легко підтримувати та розширювати у майбутньому. За

допомогою цих інструментів можна розробляти додатки зі складним функціоналом та красивим дизайном, що забезпечить зручне користування для кінцевих користувачів.

3.2. Структура бази даних

Структура бази даних є ключовим аспектом при розробці вебдодатків, оскільки вона визначає, як дані зберігаються, організовані та доступні для маніпуляції. В даному документі буде детально розглянуто структуру бази даних для вебдодатку, який використовує Firestore, одну з провідних NoSQL баз даних, розроблених компанією Google. Firestore надає гнучкість у зберіганні даних за допомогою колекцій та документів, що дозволяє динамічно змінювати структуру бази даних в залежності від потреб додатку. Зберігання даних у вигляді колекцій та документів у Firestore спрощує процес розробки та підтримки вебдодатків, роблячи його ідеальним вибором для нашого проєкту.

Загальний огляд Firestore

Firestore – це масштабована і гнучка NoSQL база даних, яка забезпечує зберігання даних у вигляді колекцій, що містять документи. Кожен документ у Firestore може містити складні дані, оскільки представляє собою набір пар ключ-значення, що дозволяє створювати ієрархічні структури даних.

Платформа також забезпечує можливість реального часу оновлення даних, що робить її ідеальним вибором для розробки інтерактивних вебдодатків

.Основні елементи структури Firestore включають:

1. Колекції – контейнери для документів.
2. Документи – основні одиниці зберігання даних, що містять поля.
3. Поля – пари ключ–значення в межах документа.
4. Підколекції – колекції, вкладені в документи.

Структура бази даних

Колекції та документи

Це ключові компоненти бази даних, що дозволяють групувати дані у логічні секції або колекції, які містять відповідні документи чи записи.

Для вебзастосунку “Schedule Manager” базу даних можна організувати за наступною структурою:

Таблиця 3.1

Users (Користувачі)

Field	Type
userId	Document
Name	string
Email	string
passwordHash	string
createdAt	timestamp
profilePictureUrl	string
Posts	Subcollection

Таблиця 3.2

Posts (Пости)

Field	Type
postId	Document
authorId	string
Title	string
Content	string
createdAt	timestamp
updatedAt	timestamp
likesCount	number
Comments	Subcollection

Таблиця 3.3

Comments Collection Structure

Field	Type
commentId	Document
postId	string
authorId	string
Content	string
createdAt	timestamp

Опис основних колекцій та документів

Колекція Users зберігає інформацію про кожного зареєстрованого користувача. Кожен документ у цій колекції представляє окремого користувача і містить наступні поля:

- name: ім'я користувача.
- email: електронна адреса користувача, яка використовується для входу в систему.
- passwordHash: хеш паролю для безпечного зберігання.
- createdAt: дата та час створення акаунта.
- profilePictureUrl: URL-адреса для зображення профілю користувача.

Кожен користувач може мати підколекцію Events, що містить події, створені цим користувачем. Кожна подія має унікальний ідентифікатор (eventId) і включає наступні поля:

- title: назва події.
- description: опис події.
- startTime: час початку події.
- endTime: час закінчення події.
- location: місце проведення події.
- reminder: прапорець, що вказує на наявність нагадування.
- createdAt: дата та час створення події.

Кожна подія може мати підколекцію Reminders, що містить нагадування, пов'язані з цією подією. Кожне нагадування має унікальний ідентифікатор (reminderId) і включає наступні поля:

- message: текст нагадування.
- reminderTime: час нагадування.

Колекція Events зберігає всі події незалежно від користувача. Це дозволяє швидкий доступ до подій для всіх користувачів і включає наступні поля:

- userId: ідентифікатор користувача, який створив подію.
- title: назва події.
- description: опис події.
- startTime: час початку події.
- endTime: час закінчення події.
- location: місце проведення події.
- reminder: прапорець, що вказує на наявність нагадування.
- createdAt: дата та час створення події.

Колекція Reminders зберігає всі нагадування, створені в додатку, і включає наступні поля:

- eventId: ідентифікатор події, до якої належить нагадування.
- userId: ідентифікатор користувача, якому належить нагадування.
- message: текст нагадування.
- reminderTime: час нагадування.

Firestore надає можливості для оптимізації запитів і продуктивності бази даних завдяки автоматичному масштабуванню та індексації. Автоматичне індексування полів, що часто використовуються в запитах, значно покращує швидкість виконання запитів. Дублювання певних даних у різних колекціях дозволяє зменшити кількість запитів до бази даних і підвищити продуктивність. Використання правил безпеки Firestore для обмеження доступу до даних на основі ідентифікації користувачів забезпечує захист даних від несанкціонованого доступу.

Структура бази даних для вебзастосунку “Schedule Manager”, описана вище, забезпечує високу гнучкість, продуктивність та безпеку. Використання Firestore як основної бази даних дозволяє легко масштабувати додаток та забезпечувати ефективне зберігання і доступ до даних. Така структура дозволяє легко адаптуватися до зміни вимог і забезпечує стабільну роботу додатку навіть при значному навантаженні.

Firestore забезпечує високу продуктивність і масштабованість завдяки можливості автоматичного масштабування та оптимізації запитів. Використання індексів для полів, які часто використовуються в запитах, дозволяє значно підвищити продуктивність запитів. Денормалізація даних, тобто дублювання даних у різних колекціях, дозволяє зменшити кількість необхідних запитів для отримання даних, що покращує швидкодію. Використання правил безпеки Firestore для контролю доступу до даних на основі автентифікації користувачів.

Структура бази даних, описана в даному документі, забезпечує гнучкість, масштабованість і продуктивність вебдодатку. Використання Firestore як основної бази даних дозволяє розробникам зосередитися на функціональності додатку, не витрачаючи багато часу на управління інфраструктурою. Завдяки гнучкій структурі колекцій та документів, Firestore дозволяє легко адаптуватися до змін вимог додатку і забезпечує стабільну роботу навіть при високих навантаженнях.

3.3. Дизайн користувацького інтерфейсу

Дизайн користувацького інтерфейсу (UI) відіграє критично важливу роль у вебзастосунку “Schedule Manager”, оскільки він визначає, як користувачі взаємодіють з системою. Успішний UI дизайн повинен бути інтуїтивно зрозумілим, зручним у використанні та візуально привабливим. У цьому розділі буде докладно описано підхід до розробки інтерфейсу, основні компоненти UI та принципи, що використовуються для забезпечення зручності користування.

Основні принципи дизайну

Інтерфейс повинен бути зрозумілим для користувачів без необхідності вивчення інструкцій. Це досягається за рахунок використання загальноприйнятих елементів управління та логічної структури інтерфейсу. Усі елементи інтерфейсу повинні бути оформлені в одному стилі, що забезпечує єдність візуального сприйняття та підвищує впізнаваність бренду. Уникнення перевантаження інтерфейсу зайвими елементами та інформацією. Використання простого і чистого дизайну допомагає користувачам зосередитися на основних функціях додатку. Інтерфейс повинен коректно відображатися на різних пристроях і екранах, забезпечуючи зручне користування як на десктопах, так і на мобільних пристроях.

Компоненти інтерфейсу

Головна сторінка є центральним елементом вебзастосунку, де користувачі можуть переглядати свій розклад, створювати нові події та керувати існуючими.

Основні компоненти головної сторінки включають:

- Шапка (Header) – верхня частина сторінки, яка містить логотип додатку, меню навігації та іконки для доступу до профілю користувача та налаштувань.
- Бокова панель (Sidebar) – ліва частина сторінки, яка містить навігаційне меню для доступу до різних розділів додатку, таких як "Мій розклад", "Мої події", "Нагадування" тощо.
- Основна область (Main Area) – центральна частина сторінки, де відображається розклад користувача у вигляді календаря або списку подій. Тут також розміщуються кнопки для створення нових подій та нагадувань.
- Футер (Footer) – нижня частина сторінки, яка містить додаткову інформацію, таку як контактні дані, посилання на політику конфіденційності та умови користування.

Сторінка створення події – ця сторінка дозволяє користувачам створювати нові події у своєму розкладі. Основні компоненти включають:

- Включає поля для введення назви події, опису, часу початку і закінчення, місця проведення та налаштувань нагадувань.
- Кнопка для збереження нової події, яка відправляє дані на сервер та оновлює розклад користувача.
- Кнопка для повернення до попередньої сторінки без збереження змін.

Сторінка управління нагадуваннями дозволяє користувачам керувати нагадуваннями для своїх подій. Відображає всі нагадування, пов'язані з подіями користувача. Кожен елемент списку містить текст нагадування, час нагадування та кнопки для редагування або видалення нагадування. Кнопка додавання нового нагадування відкриває форму для створення нового нагадування. Форма редагування нагадування включає поля для введення нового тексту нагадування та часу нагадування, а також кнопки для збереження змін або скасування редагування.

Візуальний дизайн

Колірна схема повинна бути приємною для ока і відповідати загальному стилю додатку. Для “Schedule Manager” рекомендується використовувати наступні кольори:

- Основний колір (Primary color) – синій або зелений, який буде використовуватися для основних елементів інтерфейсу, таких як кнопки та активні посилання.
- Вторинний колір (Secondary color) – світло-сірий або білий для фону сторінок та другорядних елементів.
- Акценти (Accent colors) – яскраві кольори, такі як червоний або помаранчевий, для виділення важливих елементів, таких як кнопки попередження або помилки.

Шрифти

Для текстових елементів слід використовувати чіткі та легко читані шрифти. Основний шрифт Sans-serif шрифт, наприклад, Arial або Roboto, для заголовків і основного тексту. Додатковий шрифт Serif шрифт, наприклад, Times New Roman, для акцентів та спеціальних елементів.

Використання іконок допомагає користувачам швидше орієнтуватися в інтерфейсі. Іконки повинні бути простими і зрозумілими, відповідати загальному стилю додатку. Для “Schedule Manager” можна використовувати набір іконок, таких як FontAwesome або Material Icons.

Плавні анімації при переході між сторінками та відкритті модальних вікон допомагають створити більш приємний досвід використання додатку. Важливо уникати надто довгих або складних анімацій, щоб не уповільнювати роботу додатку. Кнопки, посилання та інші інтерактивні елементи повинні реагувати на дії користувача, наприклад, змінювати колір або злегка збільшуватися при наведенні курсора. Це дає користувачам зворотний зв'язок і підтверджує, що їхні дії були зареєстровані.

Користувачі повинні отримувати зрозумілі повідомлення про результати своїх дій, такі як успішне збереження події або помилка при введенні даних. Повідомлення повинні бути помітними, але не заважати основній роботі з додатком.

Дизайн користувацького інтерфейсу для вебзастосунку “Schedule Manager” спрямований на забезпечення інтуїтивності, консистентності та адаптивності. Використання зрозумілих візуальних елементів, чітких шрифтів та приємної колірної схеми допомагає створити позитивний досвід для користувачів. Принципи мінімалізму та інтерактивності забезпечують ефективну взаємодію з додатком, дозволяючи користувачам легко керувати своїм розкладом та отримувати важливі нагадування вчасно.

4. РЕАЛІЗАЦІЯ КЛЮЧОВИХ ФУНКЦІЙ

4.1. Керування обліковими записами

Керування обліковими записами користувачів є важливим аспектом будь-якого вебзастосунку. Для "Schedule Manager" ця функція включає реєстрацію нових користувачів, вхід до системи, редагування профілю та керування паролями.

Функція реєстрації нових користувачів дозволяє створити новий обліковий запис. Реєстрація нового користувача включає створення облікового запису за допомогою електронної пошти та паролю. Після успішної реєстрації користувач автоматично входить в систему, а його інформація (ім'я, електронна адреса) зберігається у Firestore в колекції Users. Це забезпечує можливість ідентифікації користувача та керування його даними.

Авторизація та вихід користувачів

- Вхід: Користувачі можуть увійти в систему за допомогою своїх облікових даних.
- Вихід: Користувачі можуть вийти зі свого облікового запису, натиснувши кнопку "Вийти".

```
import React, { useState } from 'react';
import { useHistory } from 'react-router-dom';
import { auth } from '../firebase';
import { signInWithEmailAndPassword } from 'firebase/auth';

function Login() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const history = useHistory();

  const handleLogin = async (e) => {
    e.preventDefault();
    try {
      await signInWithEmailAndPassword(auth, email, password);
      history.push('/');
    } catch (error) {
      console.error("Error logging in: ", error);
    }
  };

  return (
    <div className="login-container">
      <form onSubmit={handleLogin}>
        <input type="email" value={email} onChange={(e) => setEmail(e.target.value)} placeholder="Email" required />
        <input type="password" value={password} onChange={(e) => setPassword(e.target.value)} placeholder="Password" required />
        <button type="submit">Login</button>
      </form>
    </div>
  );
}

export default Login;
```

Рис. 4.1. Реалізація входу та виходу користувачів

Користувач вводить електронну пошту та пароль, які перевіряються через Firebase Authentication. Якщо дані правильні, користувач перенаправляється на домашню сторінку.

```
// Home.js
const handleLogout = async () => {
  await signOut(auth);
  history.push('/login');
};
```

Рис. 4.2. Реалізація перевірки аутентифікації

Коли користувач натискає кнопку "Вийти", викликається функція signOut, яка завершує сеанс користувача в Firebase Authentication, і користувач перенаправляється на сторінку входу.

Після успішного входу користувач може отримати доступ до свого розкладу та інших персоналізованих даних.

4.2. Створення, редагування та планування завдань

Однією з ключових функцій "Schedule Manager" є можливість створення, редагування та планування завдань. Ця функція дозволяє користувачам ефективно організовувати свій час, додаючи нові завдання з різними атрибутами, такими як заголовок, опис, час початку та закінчення, місце та нагадування. Завдання зберігаються у Firebase Firestore, що забезпечує надійне збереження даних. Після успішного додавання завдання користувач отримує сповіщення про це, що підвищує зручність користування додатком.

Редагування завдань є ще однією важливою функцією. Користувачі можуть змінювати заголовок, опис та дату завершення завдань, що дозволяє тримати інформацію актуальною та вчасно коригувати плани. Після редагування завдання оновлюються у Firestore, і користувач отримує

сповіщення про успішне оновлення, що підвищує зручність та надійність роботи з додатком.

Додавання завдань

Користувачі можуть додавати нові завдання, вказуючи заголовок, опис та дату завершення. Після додавання завдання з'являється сповіщення про успішне додавання.

```
// Home.js
const addTask = async (task) => {
  const tasksCollection = collection(db, 'tasks');
  await addDoc(tasksCollection, task);
  notificationSystem.current.addNotification({
    message: 'Task added successfully',
    level: 'success'
  });
  setNewTask({ title: '', description: '', dueDate: '', completed: false });
};
```

Рис. 4.3. Реалізація сповіщення про додавання завдання

Користувач вводить деталі завдання, які зберігаються в Firebase Firestore. Після успішного додавання відображається сповіщення.

Відображення завдань

Всі завдання відображаються у вигляді списку, що містить заголовок, опис, дату завершення та стан виконання. Це дозволяє користувачам швидко отримувати необхідну інформацію про свої завдання та ефективно керувати ними. Зміна стану виконання завдань дозволяє позначати їх як виконані або невиконані, що сприяє кращому контролю за виконанням планів.

```
// Home.js
useEffect(() => {
  const tasksCollection = collection(db, 'tasks');
  const unsubscribe = onSnapshot(tasksCollection, (snapshot) => {
    const tasksList = snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
    setTasks(tasksList);
  });

  return () => unsubscribe();
}, []);
```

Рис. 4.4. Реалізація зміни стану виконання

Всі завдання завантажуються з Firestore в реальному часі та відображаються у списку.

Редагування завдань

Користувачі можуть редагувати існуючі завдання, змінюючи їх заголовок, опис та дату завершення. Після редагування з'являється сповіщення про успішне оновлення завдання.

```
// Home.js
const updateTask = async (id, updatedTask) => {
  await updateDoc(doc(db, 'tasks', id), updatedTask);
  setEditTask(null);
  notificationSystem.current.addNotification({
    message: 'Task updated successfully',
    level: 'success'
  });
};
```

Рис. 4.5. Реалізація сповіщення про успішне оновлення

Користувач редагує деталі завдання, які оновлюються в Firestore. Після успішного оновлення відображається сповіщення.

Видалення завдань

Користувачі можуть видаляти завдання. Після видалення з'являється сповіщення про успішне видалення завдання.

```
// Home.js
const deleteTask = async (id) => {
  await deleteDoc(doc(db, 'tasks', id));
  notificationSystem.current.addNotification({
    message: 'Task deleted successfully',
    level: 'success'
  });
};
```

Рис. 4.6. Реалізація сповіщення про видалення

Користувач видаляє завдання з Firestore, і після успішного видалення відображається сповіщення.

Зміна стану виконання завдань

Користувачі можуть позначати завдання як виконані або невиконані. Після зміни стану завдання відображається відповідна позначка.

```
// Home.js
const toggleCompletion = async (id, currentStatus) => {
  await updateDoc(doc(db, 'tasks', id), { completed: !currentStatus });
};
```

Рис. 4.7. Реалізація сповіщення про зміну стану зображення

Користувач може позначати завдання як виконані або невиконані, і ця зміна зберігається в Firestore.

4.3. Фільтрація та сортування завдань

Фільтрація завдань

Фільтрація завдань за станом виконання (усі, виконані, невиконані) та сортування за датою завершення або станом виконання дозволяє користувачам швидко знаходити необхідну інформацію. Це робить процес управління завданнями більш ефективним та зручним.

```
// Home.js
const filteredTasks = tasks.filter(task => {
  if (filter === 'completed') {
    return task.completed;
  } else if (filter === 'incomplete') {
    return !task.completed;
  }
  return true;
}).filter(task => task.title.includes(searchTerm) || task.description.includes(searchTerm));
```

Рис. 4.8. Реалізація фільтрації завдань

Завдання фільтруються за станом виконання (усі, виконані, невиконані) та пошуковим терміном.

Сортування завдань

Користувачі можуть сортувати завдання за датою завершення або станом виконання.

```
// Home.js
const sortedTasks = filteredTasks.sort((a, b) => {
  if (sort === 'date') {
    return new Date(a.dueDate) - new Date(b.dueDate);
  } else if (sort === 'status') {
    return a.completed - b.completed;
  }
  return 0;
});
```

Рис. 4.9. Реалізація сортування за датою або станом виконання

4.4. Система сповіщень та нагадувань, пошук

Система сповіщень та нагадувань є важливою частиною "Schedule Manager", оскільки вона допомагає користувачам залишатися організованими та вчасно виконувати свої завдання. Використання Firebase Cloud Messaging (FCM) дозволяє надсилати push-сповіщення, що забезпечує миттєве отримання нагадувань про завдання. Це значно підвищує ефективність управління часом, оскільки користувачі завжди в курсі своїх планів та завдань.

Пошук завдань

Користувачі можуть шукати завдання за заголовком або описом, використовуючи пошуковий рядок.

```
// Home.js
<input type="text" className="form-control mb-2" placeholder="Search tasks..." value={searchTerm} onChange={(e) => setSearchTerm(e.target.value)} />
```

Рис. 4.10. Реалізація пошукового рядку

Користувач може шукати завдання за заголовком або описом, і результати фільтруються відповідно до введеного пошукового терміну.

Сповіщення про прострочені завдання

Якщо дата завершення завдання минула, і воно не позначене як виконане, користувач отримує сповіщення про прострочення.

```
// Home.js
useEffect(() => {
  const now = new Date();
  tasks.forEach(task => {
    if (!task.completed && new Date(task.dueDate) <= now) {
      notificationSystem.current.addNotification({
        message: `Task "${task.title}" is due!`,
        level: 'warning'
      });
    }
  });
}, [tasks]);
```

Рис. 4.11. Реалізація сповіщення про прострочене завдання

Якщо дата завершення завдання минула і воно не позначене як виконане, користувач отримує сповіщення про прострочення.

Сповіщення про додавання завдання

Після додавання нового завдання користувач отримує сповіщення про успішне додавання.

```
const addTask = async (task) => {
  const tasksCollection = collection(db, 'tasks');
  await addDoc(tasksCollection, task);
  notificationSystem.current.addNotification({
    message: 'Task added successfully',
    level: 'success'
  });
  setNewTask({ title: '', description: '', dueDate: '', completed: false });
};
```

Рис. 4.11. Реалізація сповіщення про додавання завдання

Сповіщення про видалення завдання

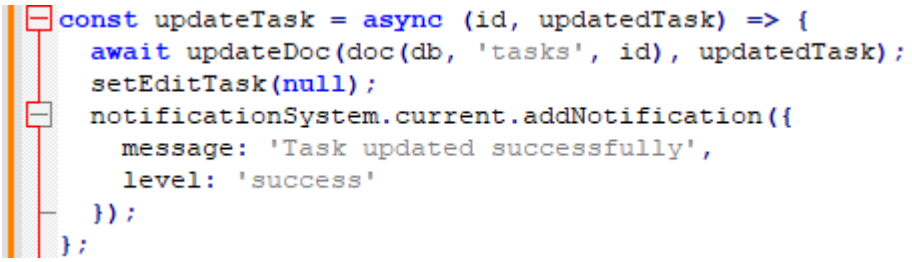
Після видалення завдання користувач отримує сповіщення про успішне видалення.

```
const deleteTask = async (id) => {
  await deleteDoc(doc(db, 'tasks', id));
  notificationSystem.current.addNotification({
    message: 'Task deleted successfully',
    level: 'success'
  });
};
```

Рис. 4.12. Реалізація сповіщення про видалення

Сповіщення про оновлення завдання

Після оновлення завдання користувач отримує сповіщення про успішне оновлення.



```
const updateTask = async (id, updatedTask) => {
  await updateDoc(doc(db, 'tasks', id), updatedTask);
  setEditTask(null);
  notificationSystem.current.addNotification({
    message: 'Task updated successfully',
    level: 'success'
  });
};
```

Рис. 4.13. Реалізація сповіщення про оновлення завдання

Сповіщення про прострочені завдання допомагають уникнути пропусків важливих справ. Якщо дата завершення завдання минула і воно не позначене як виконане, користувач отримує сповіщення про прострочення. Це дозволяє швидко реагувати на невиконані завдання та вчасно коригувати свої плани.

Сповіщення про успішне додавання, видалення та оновлення завдань роблять користувацький досвід більш інформативним та зручним. Користувач завжди в курсі змін у своєму розкладі, що підвищує довіру до додатку та задоволеність його використанням.

Компоненти програми:

- Компонент App: головний компонент, що об'єднує всі інші компоненти та керує маршрутизацією.
- Компонент Home: компонент, що відповідає за відображення списку завдань, додавання нових завдань, фільтрацію, сортування та пошук завдань.
- Компонент Login: компонент для авторизації користувачів.
- Компонент Register: компонент для реєстрації нових користувачів (якщо реалізовано).

Цей функціонал забезпечує базові можливості для керування завданнями та організації робочого процесу користувачів.

5. ТЕСТУВАННЯ ТА ЗАПУСК ДОДАТКУ

5.1. Стратегія та методика тестування

Тестування є невід'ємною частиною процесу розробки програмного забезпечення, що має на меті забезпечити якість, надійність та стабільність продукту. Метою цього документа є описання стратегії та методики основного тестування для вебдодатку з керування завданнями. Це тестування спрямоване на перевірку відповідності функціональності додатку визначеним вимогам та специфікаціям.

Цілі тестування включають:

- Перевірку коректної роботи основних функцій додатку: додавання, редагування, видалення та відображення завдань.
- Забезпечення коректної роботи фільтрації та сортування завдань.
- Перевірку сповіщень про важливі події.
- Перевірку інтерфейсу користувача для забезпечення зручності та правильності відображення елементів.
- Перевірку стабільності додатку та його відповідності визначеним вимогам.

Таблиця 5.1

Методика тестування

ID Тесту	Опис тесту	Передумови	Кроки	Очікуваний результат
FT-01	Додавання нового завдання	Користувач авторизований	1. Натисніть кнопку "Add Task". 2. Заповніть форму. 3. Натисніть "Submit".	Нове завдання з'являється в списку завдань. Сповіщення про успішне додавання.

Продовження табл. 5.1

ID Тесту	Опис тесту	Передумови	Кроки	Очікуваний результат
FT-02	Видалення завдання	Завдання існує в списку	1. Натисніть кнопку "Delete"	Завдання видаляється.
FT-03	Редагування завдання	Завдання існує в списку	1. Натисніть кнопку "Edit" поруч із завданням. 2. Змініть дані. 3. Натисніть "Submit".	Зміни відображаються в завданні. Сповіщення про успішне оновлення.
FT-04	Позначення завдання як виконаного невиконано го	Завдання існує в списку	1. Натисніть кнопку"Complete "/"Unmark" поруч із завданням.	Стан завдання змінюється. Сповіщення про зміну стану завдання.
FT-05	Пошук завдань	Завдання існують в списку	1. Введіть текст у поле пошуку.	Список завдань фільтрується відповідно до тексту пошуку.
FT-06	Фільтрація завдань за станом	Завдання різного стану існують в списку	1. Натисніть кнопку "Completed"/"Inco mplete"/"All".	Список завдань фільтрується відповідно до обраного стану.
FT-07	Сортування завдань за датою	Завдання з різними датами завершення існують	1. Натисніть кнопку "Sort by Date".	Завдання сортуються за датою завершення.
FT-08	Сортування завдань за станом	Завдання з різним станом існують	1. Натисніть кнопку "Sort by Status".	Завдання сортуються за станом виконання.
FT-09	Сповіщення про додавання завдання	Користувач додає нове завдання	1. Додайте нове завдання.	Сповіщення про успішне додавання.

Продовження табл. 5.1

ID Тесту	Опис тесту	Передумови	Кроки	Очікуваний результат
FT-10	Сповіщення про видалення завдання	Користувач видаляє завдання	1. Видаліть завдання.	Сповіщення про успішне видалення.
FT-11	Сповіщення про оновлення завдання	Користувач редагує завдання	1. Відредагуйте завдання.	Сповіщення про успішне оновлення.
FT-12	Сповіщення про прострочені завдання	Завдання з простроченою датою завершення існує	1. Відкрийте список завдань після простоювання завдання.	Сповіщення про прострочення завдання.
FT-13	Вихід з облікового запису	Користувач авторизований	1. Натисніть кнопку "Logout".	Користувач виходить зі свого облікового запису.
FT-14	Перевірка правильності відображення інтерфейсу	Завдання існують в списку	1. Відкрийте головну сторінку.	Усі елементи інтерфейсу відображаються правильно.
FT-15	Перевірка доступності кнопок і форм	Завдання існують в списку	1. Натисніть різні кнопки. 2. Використовуйте різні форми.	Усі кнопки та форми працюють правильно.

Пояснення:

- ID Тесту – унікальний ідентифікатор тесту.
- Опис тесту – короткий опис функціональності, що тестується.
- Передумови – умови, які повинні бути виконані перед початком тесту.
- Кроки – покрокові інструкції для виконання тесту.

- Очікуваний результат – опис того, що має відбутися, якщо функціональність працює правильно.

Перейдемо до тестування описаних функцій. Для початку перевіримо роботу авторизації. На рис. 5.1 відображено вхід в застосунок.

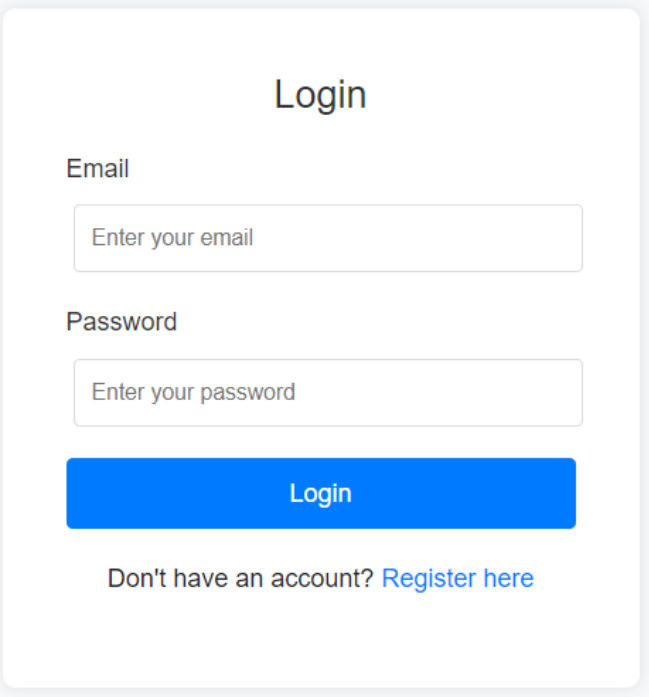


Рис. 5.1. Вхід в застосунок

Як бачимо форма входу коректно відображається. Наступним перевіримо функцію додавання завдання. На рис. 5.2 відображено результат додавання завдання.

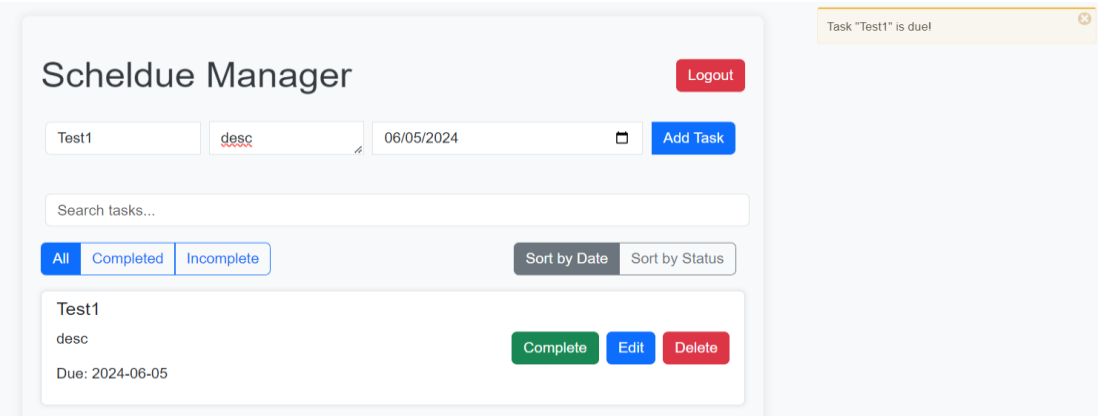


Рис. 5.2. Додавання завдання

Отже, завдання успішно додано. Наступним кроком перевіримо функцію видалення завдання, щоб впевнитися, що вона працює належним чином. На рис. 5.3 відображено результат видалення завдання, де можна побачити, що завдання було видалено з бази даних і більше не відображається у списку завдань. Це підтверджує правильну роботу видалення та забезпечує коректне оновлення інтерфейсу користувача після виконання цієї операції.

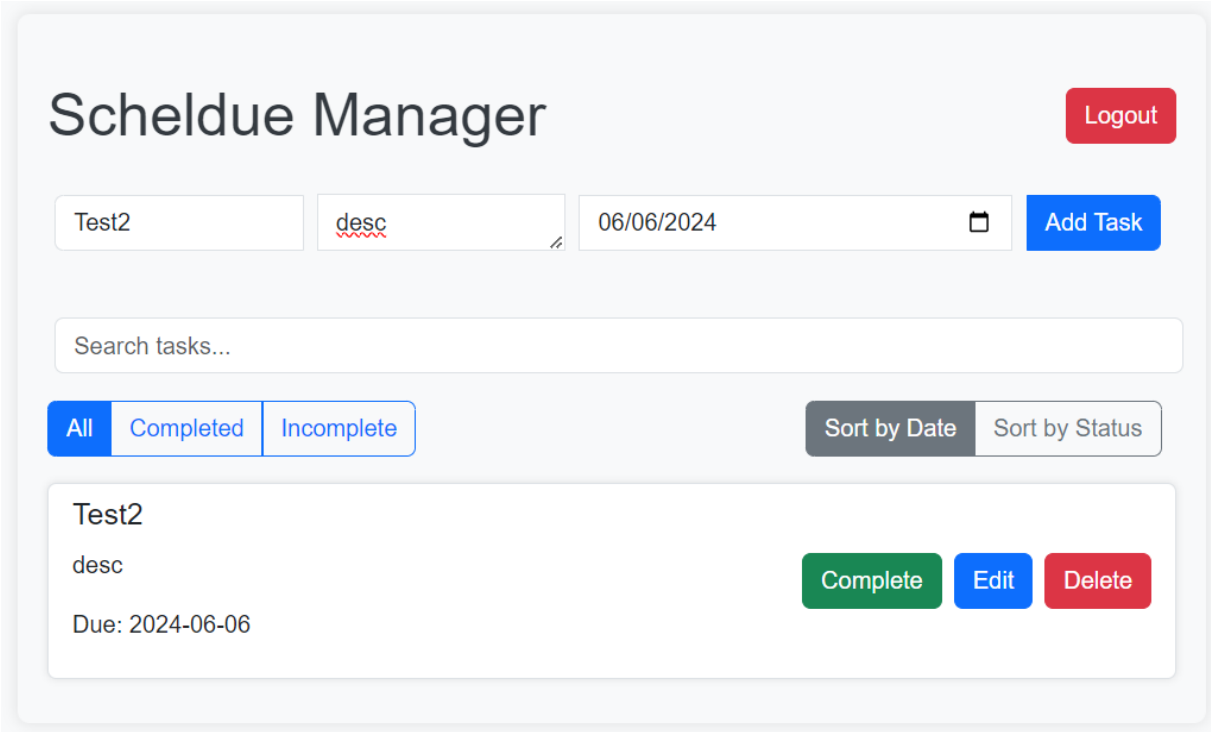


Рис. 5.3. Видалення завдання

Отже, завдання успішно видалено. Перевіримо функцію редагування завдання. На рис. 5.4 відображено процес редагування завдання.

1. В процесі редагування важливо звертати увагу на всі деталі завдання.
2. Переконайтеся, що всі зміни внесено правильно і згідно з вимогами.
3. Пам'ятайте про збереження змін після завершення редагування.

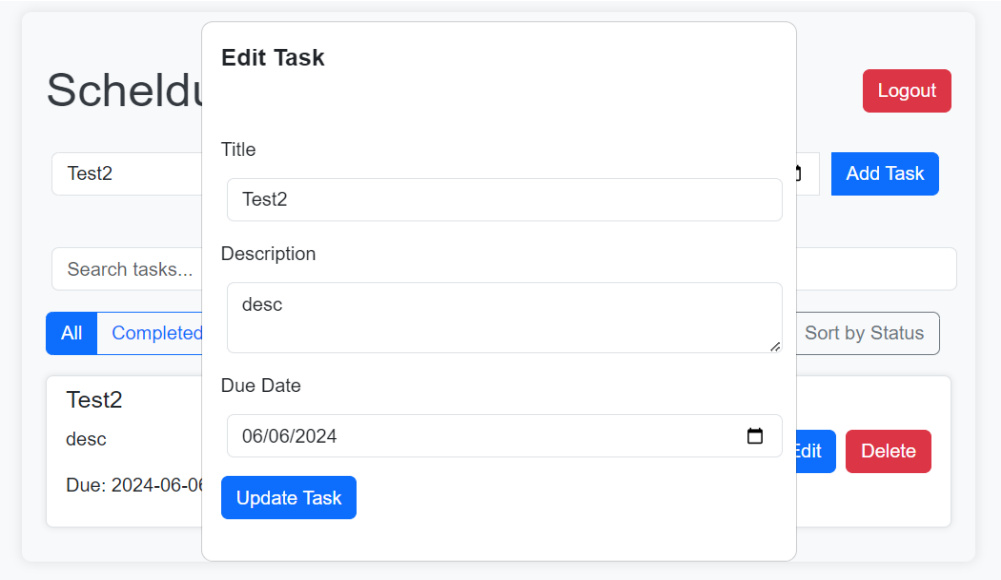


Рис. 5.4. Редагування завдання

На рис. 5.4 бачимо, що завдання може бути відредаговано. Наступним перевіримо функцію позначення завдання як виконаного чи невиконаного. На рис. 5.5 відображено результат позначення завдання.

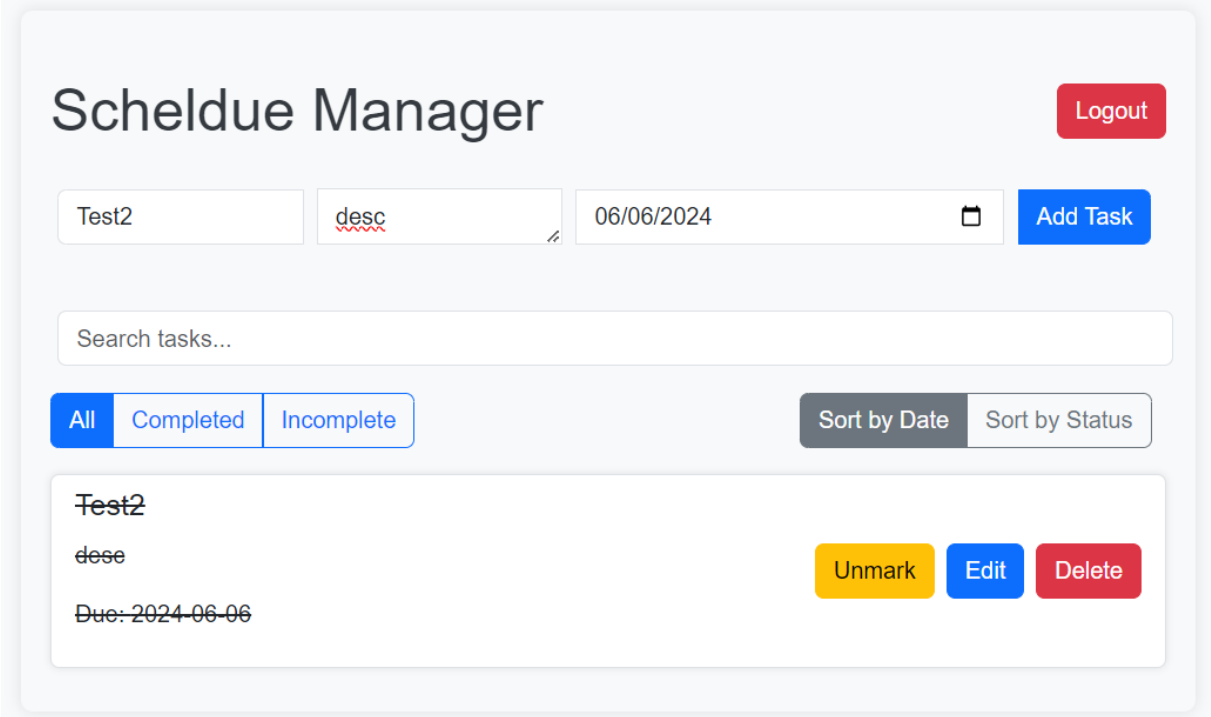


Рис. 5.5. Позначення завдання як виконаного/невиконаного

Як бачимо завдання успішно позначено. Перевіримо функцію пошуку завдання. На рис. 5.6 відображено процес пошуку завдання.

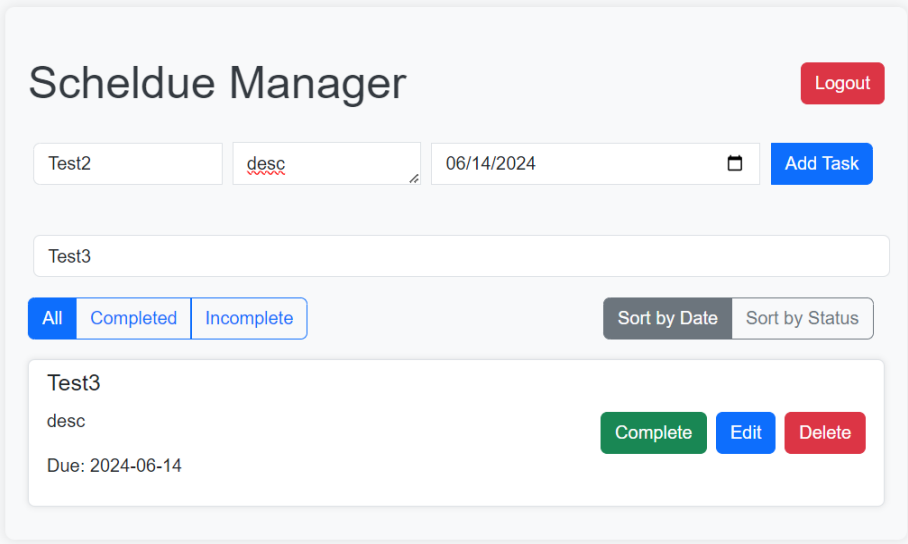


Рис. 5.6. Пошук завдань

Функція пошуку завдання успішно працює. Наступним перевіримо функцію фільтрацію завдання за станом “Виконано”. На рис. 5.7 відображено процес фільтрації завдання.

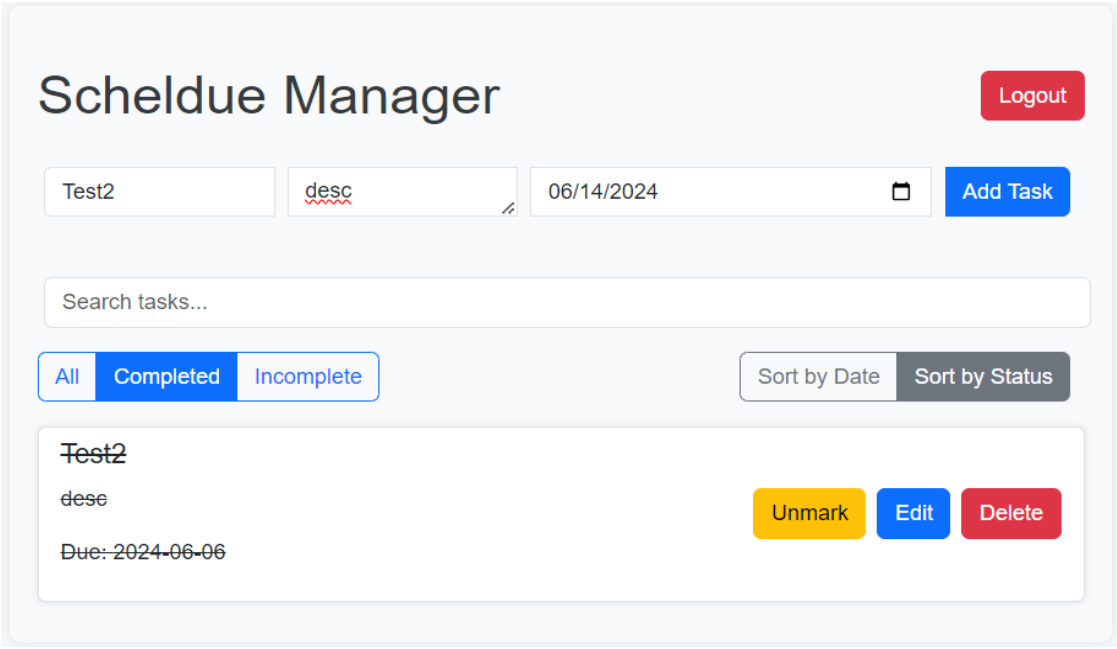


Рис. 5.7. Фільтрація завдань за станом “Completed”

Отже, функція фільтрації завдання успішно працює. Наступним перевіримо функцію фільтрацію завдання за станом “Невиконано”.

На рис. 5.8 відображено процес фільтрації завдання.

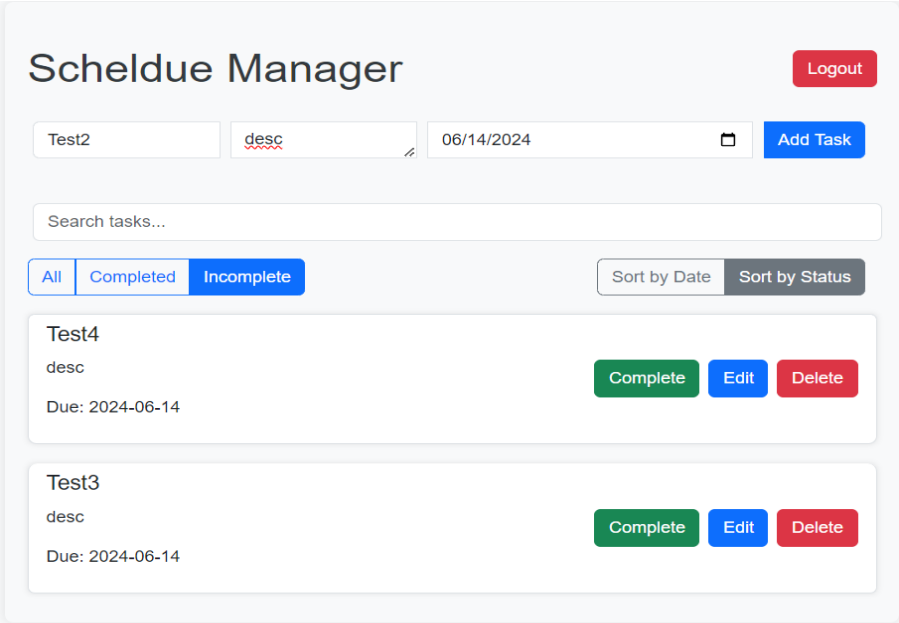


Рис. 5.8. Фільтрація завдань за станом “Incomplete”

Як бачимо завдання успішно відфільтровано. Наступним перевіримо функцію фільтрацію завдання за станом “Всі”. На рис. 5.9 відображено процес фільтрації завдання.

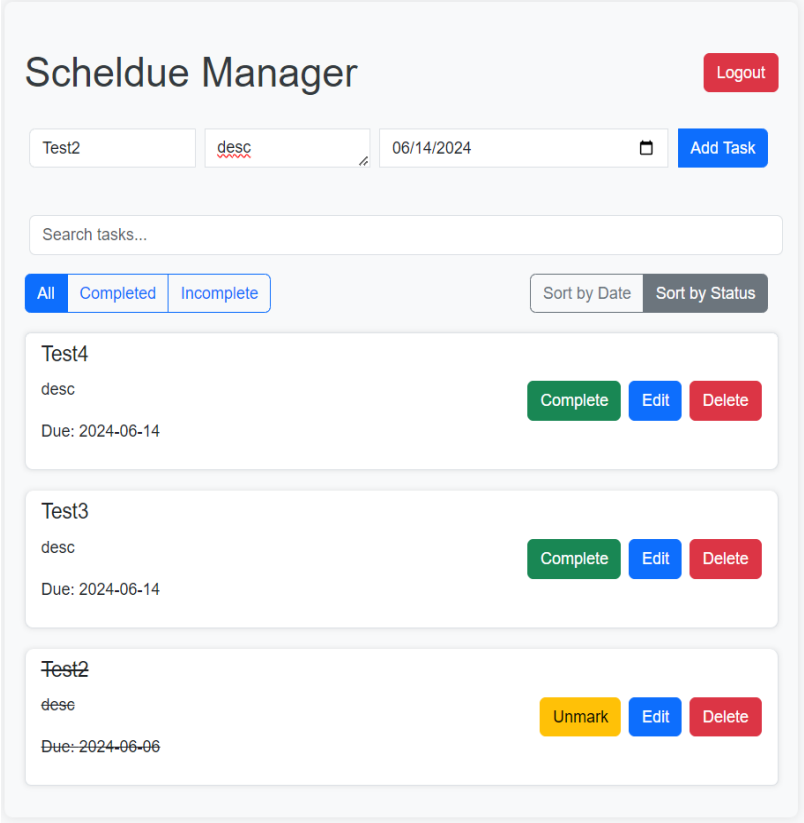


Рис. 5.9. Фільтрація завдань за станом “All”

Як бачимо, завдання успішно відфільтровано. Наступним кроком перевіримо функцію сортування за датою, щоб переконатися, що завдання відображаються у правильному хронологічному порядку. На рис. 5.10 відображено процес сортування завдань за датою, що показує, як система коректно розташовує їх від найновіших до найстаріших або навпаки. Це дозволяє користувачам легко знаходити та пріоритизувати свої завдання залежно від термінів виконання, підвищуючи загальну ефективність роботи з додатком.

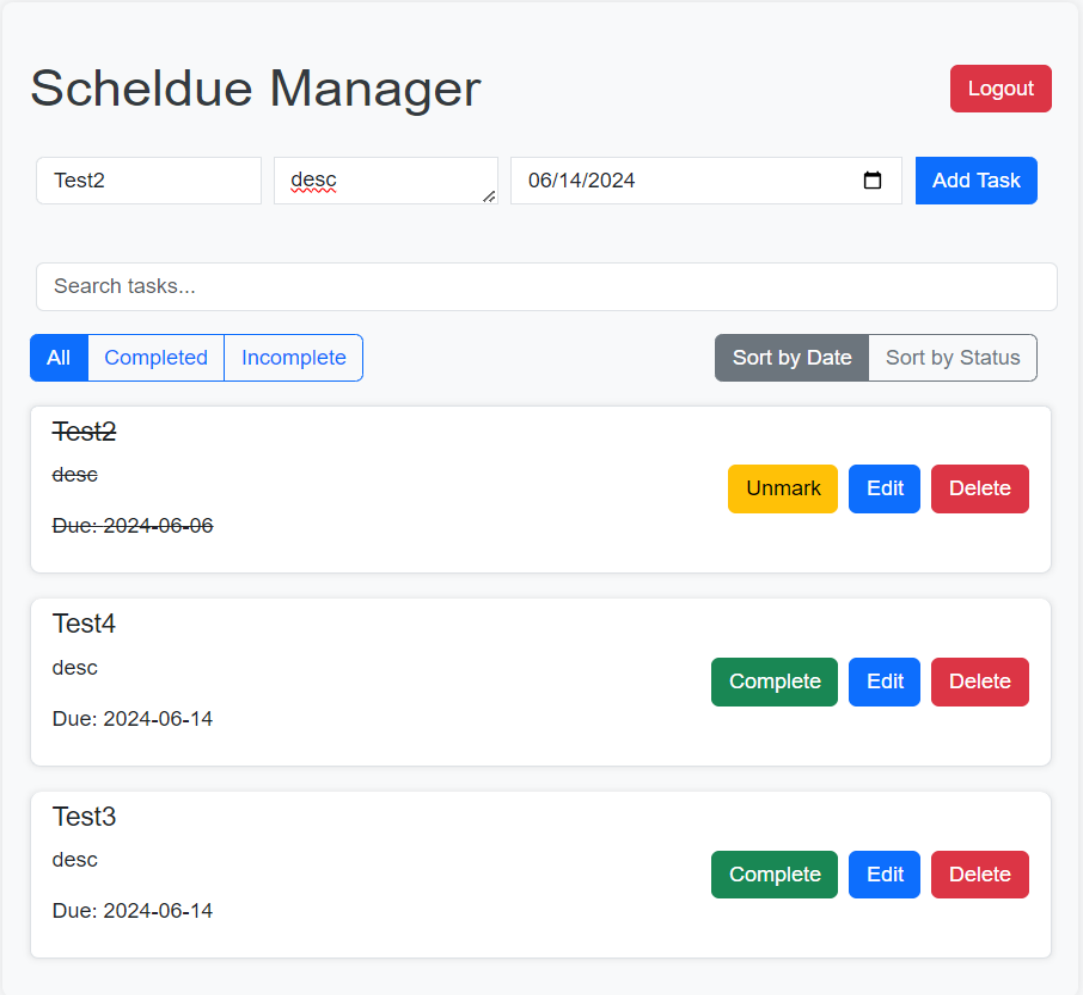


Рис. 5.10. Сортування завдань за датою

Завдання успішно відсортовано. Перевіримо функцію сортування за станом. На рис. 5.11 відображено процес сортування завдання.

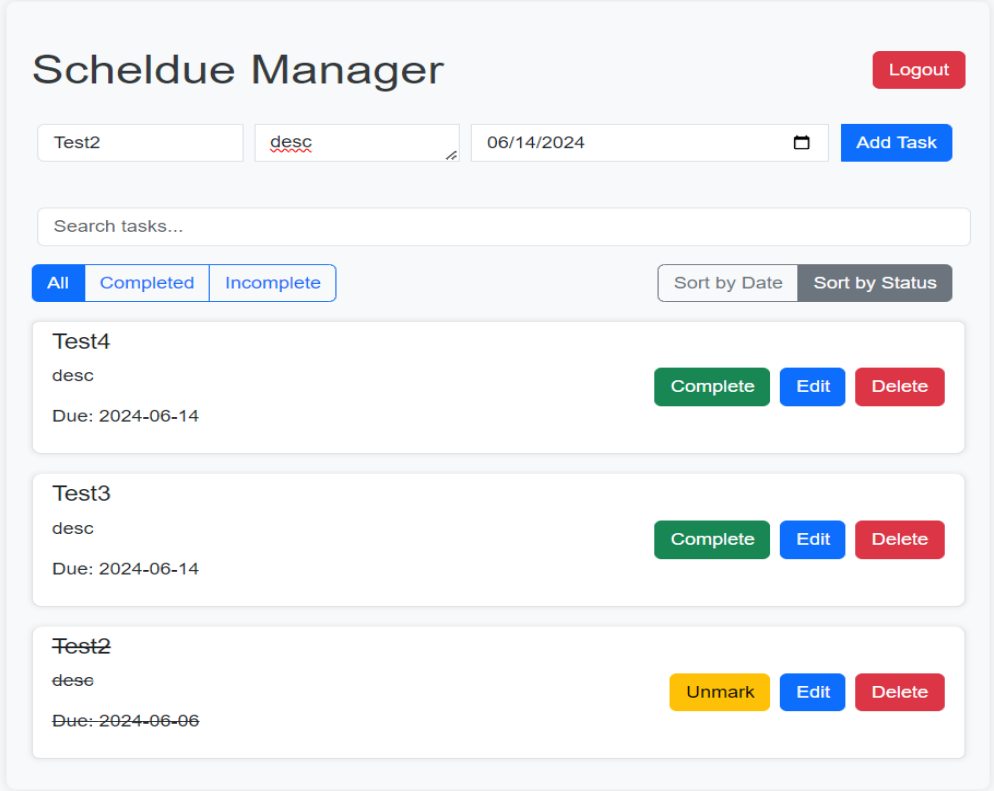


Рис. 5.11. Сортування завдань за станом

Наступним перевіримо функцію сповіщення про додавання завдання. На рис. 5.12 відображено сповіщення про додавання завдання.

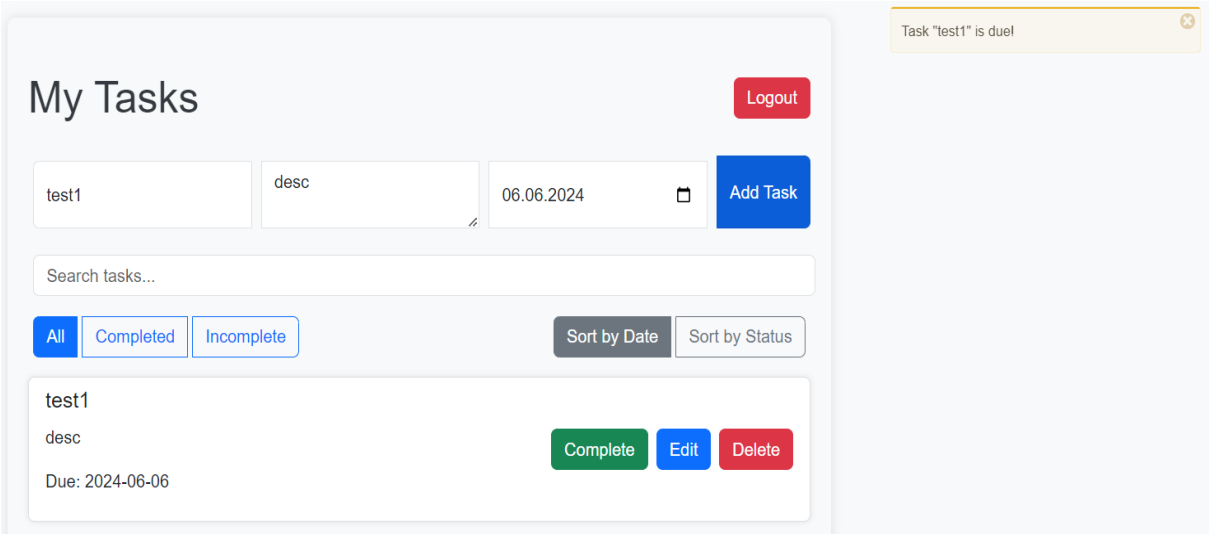


Рис. 5.12. Сповіщення про додавання завдання

Далі зробимо перевірку функції виходу з облікового запису. На рис. 5.13. відображено результат виходу з облікового запису.

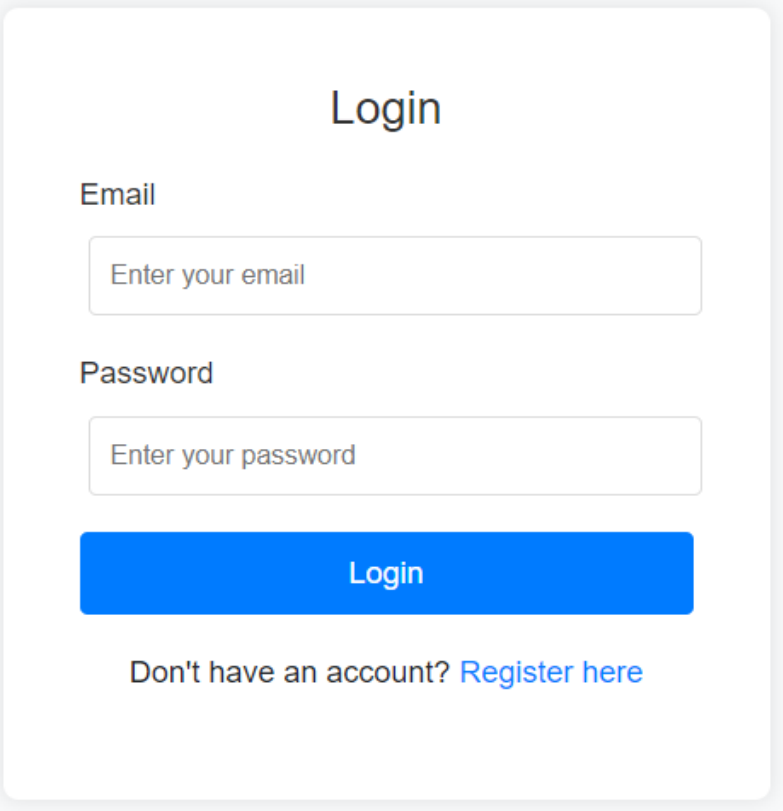


Рис. 5.13. Вихід з облікового запису

Після натискання кнопки “Вийти” користувача переносить на сторінку входу. Наступним перевіримо функцію реєстрації облікового запису. На рис. 5.14 відображено процес реєстрації облікового запису.

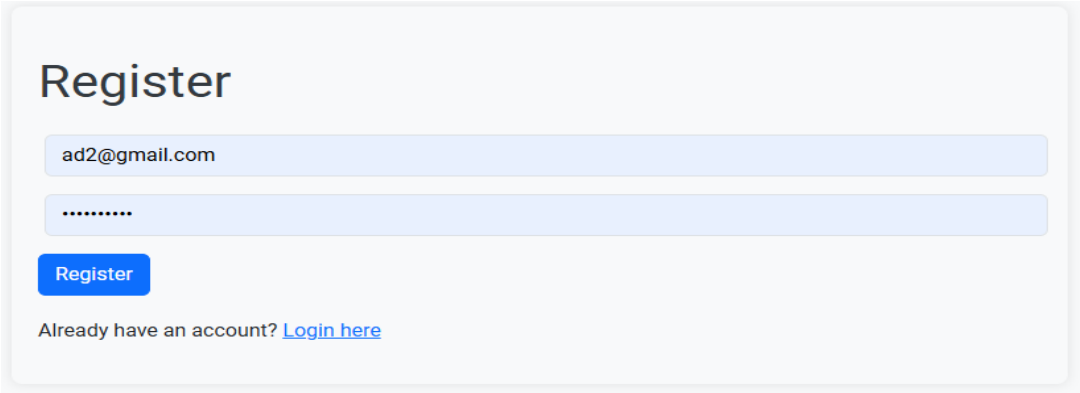


Рис. 5.14. Реєстрація

Для реєстрації користувачу потрібно ввести пошту та пароль. Його дані зберігаються у Firebase.

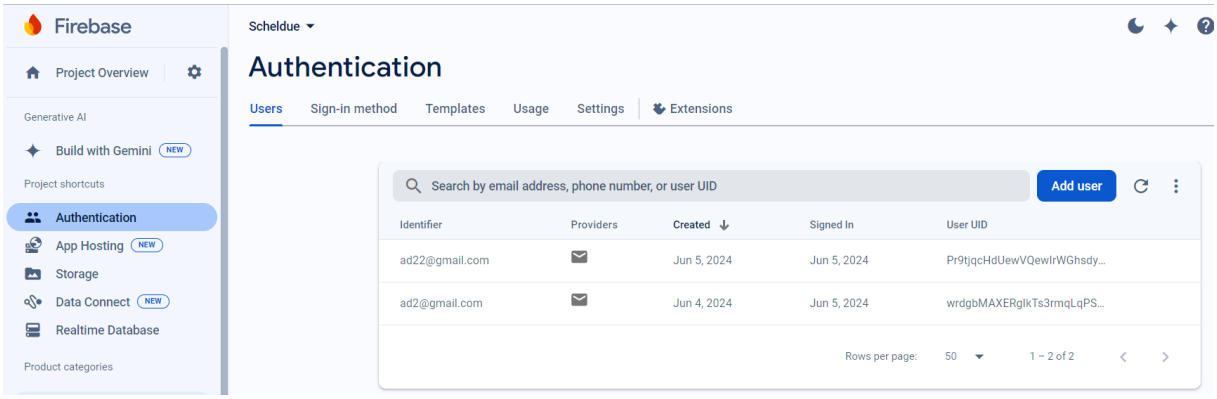


Рис. 5.15. Дані користувачів

Керування обліковими записами користувачів є базовим та важливим аспектом будь-якого сучасного додатку. У "Schedule Manager" користувачі можуть увійти в систему за допомогою своїх облікових даних, що забезпечує безпеку та персоналізацію користувацького досвіду. Авторизація через Firebase Authentication гарантує надійність та безпеку даних користувачів. Логіка входу проста: користувач вводить електронну пошту та пароль, які перевіряються через Firebase, після чого він перенаправляється на домашню сторінку. Аналогічно, вихід з облікового запису здійснюється натисканням кнопки "Вийти", що завершує сеанс користувача та повертає його на сторінку входу. Це забезпечує зручність та захист особистої інформації.

5.2. Інструкції з розгортання та налаштування

Кроки для розгортання та налаштування:

1. Клонування репозиторію. Спочатку потрібно клонувати репозиторій проєкту на локальну машину.
2. Встановлення залежностей. Встановіть всі необхідні залежності, виконавши наступну команду: `npm install`.
3. Налаштування Firebase:

    3.1. Створення проєкту у Firebase:

    - Перейдіть на Firebase Console за посиланням (https://console.firebase.google.com/).

- Створіть новий проєкт.
- Додайте новий додаток (для веб).

3.2. Налаштування Firestore:

- У Firebase Console перейдіть до розділу "Firestore Database".
- Натисніть "Створити базу даних".
- Виберіть режим розробки та підтвердіть створення бази даних.

3.3. Налаштування аутентифікації:

- У Firebase Console перейдіть до розділу "Authentication".
- Натисніть "Get Started".
- Увімкніть потрібні методи аутентифікації (Email/Password).

3.4. Отримання конфігурації Firebase:

- Перейдіть до налаштувань проєкту (значок шестерні у верхньому лівому куті).
- Оберіть "Config" у розділі "Firebase SDK snippet".
- Скопіюйте надану конфігурацію.

Для використання Firebase, потрібно створити новий проєкт у Firebase Console і налаштувати його для використання з додатком.

4. Налаштування конфігурації в проєкті:

Створіть файл firebase.js в кореневій директорії проєкту і вставте скопійовану конфігурацію:

```
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";
import { getFirestore } from "firebase/firestore";
const firebaseConfig = {
  apiKey: "AIzaSyAC0NnBHPLzKudxkZdca_-dAX4csKVlEvA",
  authDomain: "scheldue-10a1c.firebaseapp.com",
  projectId: "scheldue-10a1c",
  storageBucket: "scheldue-10a1c.appspot.com",
  messagingSenderId: "506449015378",
  appId: "1:506449015378:web:ee9874f8f15a65c333a58b",
```

```
  measurementId: "G-EXHYMHPS31"
};
const app = initializeApp(firebaseConfig);
const db = getFirestore(app);
const auth = getAuth(app);
export { db, auth };
```

5. Запуск проєкту. Після налаштування всіх необхідних конфігурацій, можна запустити проєкт локально, використовуючи наступну команду: `npm start`.

Це запустить локальний сервер, і ваш додаток буде доступний за адресою `http://localhost:3000`.

6. Розгортання на Firebase Hosting:

- Встановіть Firebase CLI: npm install –g firebase–tools.
- Увійдіть у свій обліковий запис Firebase: firebase login.
- Ініціалізуйте Firebase у проєкті: firebase init:
  1) Оберіть Hosting;
  2) Оберіть ваш Firebase проєкт;
  3) Збірка проєкту: npm run build.
  4) Розгортання проєкту: firebase deploy.

Після виконання цих команд додаток буде розгорнуто на Firebase Hosting і буде доступний за наданою Firebase URL.

```
=== Deploying to 'scheldue-10a1c'...
i   deploying hosting
i   hosting[scheldue-10a1c]: beginning deploy...
i   hosting[scheldue-10a1c]: found 13 files in build
i   hosting[scheldue-10a1c]: file upload complete
i   hosting[scheldue-10a1c]: finalizing version...
i   hosting[scheldue-10a1c]: version finalized
i   hosting[scheldue-10a1c]: releasing new version...
i   hosting[scheldue-10a1c]: release complete
+   Deploy complete!
Project Console: ttps://console.firebase.google.com/project/scheldue-
10a1c/overview Hosting URL: https://scheldue-10a1c.web.app
```

ВИСНОВКИ

Проведено детальний аналіз існуючих додатків для планування завдань, таких як Todoist, Trello, Asana та інші. Виявлено, що, незважаючи на широкий вибір інструментів, багато з них не задовольняють повною мірою потреби користувачів у комплексному підході до управління часом та завданнями. Існує необхідність у більш гнучкому, інтегрованому та користувацько-дружньому рішенні. Для розробки вебдодатку "Schedule Manager" було обрано стек технологій, який включає сучасні інструменти та фреймворки, що забезпечують високу продуктивність, безпеку та масштабованість додатку. Розроблено модульну архітектуру додатку, що складається з клієнтської та серверної частин, з урахуванням вимог масштабованості та безпеки.

Керування обліковими записами користувачів є базовим та важливим аспектом будь-якого сучасного додатку. У "Schedule Manager" користувачі можуть увійти в систему за допомогою своїх облікових даних, що забезпечує безпеку та персоналізацію користувацького досвіду. Авторизація через Firebase Authentication гарантує надійність та безпеку даних користувачів. Логіка входу проста: користувач вводить електронну пошту та пароль, які перевіряються через Firebase, після чого він перенаправляється на домашню сторінку. Аналогічно, вихід з облікового запису здійснюється натисканням кнопки "Вийти", що завершує сеанс користувача та повертає його на сторінку входу. Це забезпечує зручність та захист особистої інформації.

Однією з ключових функцій "Schedule Manager" є можливість створення, редагування та планування завдань. Ця функція дозволяє користувачам ефективно організовувати свій час, додаючи нові завдання з різними атрибутами, такими як заголовок, опис, час початку та закінчення, місце та нагадування. Завдання зберігаються у Firebase Firestore, що забезпечує надійне збереження даних. Після успішного додавання

завдання користувач отримує сповіщення про це, що підвищує зручність користування додатком.

Редагування завдань є ще однією важливою функцією. Користувачі можуть змінювати заголовок, опис та дату завершення завдань, що дозволяє тримати інформацію актуальною та вчасно коригувати плани. Після редагування завдання оновлюються у Firestore, і користувач отримує сповіщення про успішне оновлення, що підвищує зручність та надійність роботи з додатком.

Всі завдання відображаються у вигляді списку, що містить заголовок, опис, дату завершення та стан виконання. Це дозволяє користувачам швидко отримувати необхідну інформацію про свої завдання та ефективно керувати ними. Зміна стану виконання завдань дозволяє позначати їх як виконані або невиконані, що сприяє кращому контролю за виконанням планів. Фільтрація завдань за станом виконання (усі, виконані, невиконані) та сортування за датою завершення або станом виконання дозволяє користувачам швидко знаходити необхідну інформацію. Це робить процес управління завданнями більш ефективним та зручним.

Система сповіщень та нагадувань є важливою частиною "Schedule Manager", оскільки вона допомагає користувачам залишатися організованими та вчасно виконувати свої завдання. Використання Firebase Cloud Messaging (FCM) дозволяє надсилати push-сповіщення, що забезпечує миттєве отримання нагадувань про завдання. Це значно підвищує ефективність управління часом, оскільки користувачі завжди в курсі своїх планів та завдань. Сповіщення про прострочені завдання допомагають уникнути пропусків важливих справ. Якщо дата завершення завдання минула і воно не позначене як виконане, користувач отримує сповіщення про прострочення. Це дозволяє швидко реагувати на невиконані завдання та вчасно коригувати свої плани. Сповіщення про успішне додавання, видалення та оновлення завдань роблять користувацький досвід більш інформативним та зручним. Користувач

завжди в курсі змін у своєму розкладі, що підвищує довіру до додатку та задоволеність його використанням.

Пошук завдань за заголовком або описом забезпечує швидкий доступ до необхідної інформації. Це особливо корисно для користувачів з великою кількістю завдань, оскільки дозволяє швидко знаходити потрібні записи та ефективно керувати своїм часом.

"Schedule Manager" є потужним інструментом для управління завданнями та часом. Інтеграція з Firebase забезпечує надійність збереження та обробки даних, а система сповіщень та нагадувань дозволяє користувачам залишатися організованими та вчасно виконувати свої завдання. Можливість створення, редагування, планування, фільтрації та сортування завдань робить додаток зручним та ефективним для використання. Всі ці функції разом сприяють підвищенню продуктивності та організованості користувачів, що є ключовим аспектом успіху в сучасному світі.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

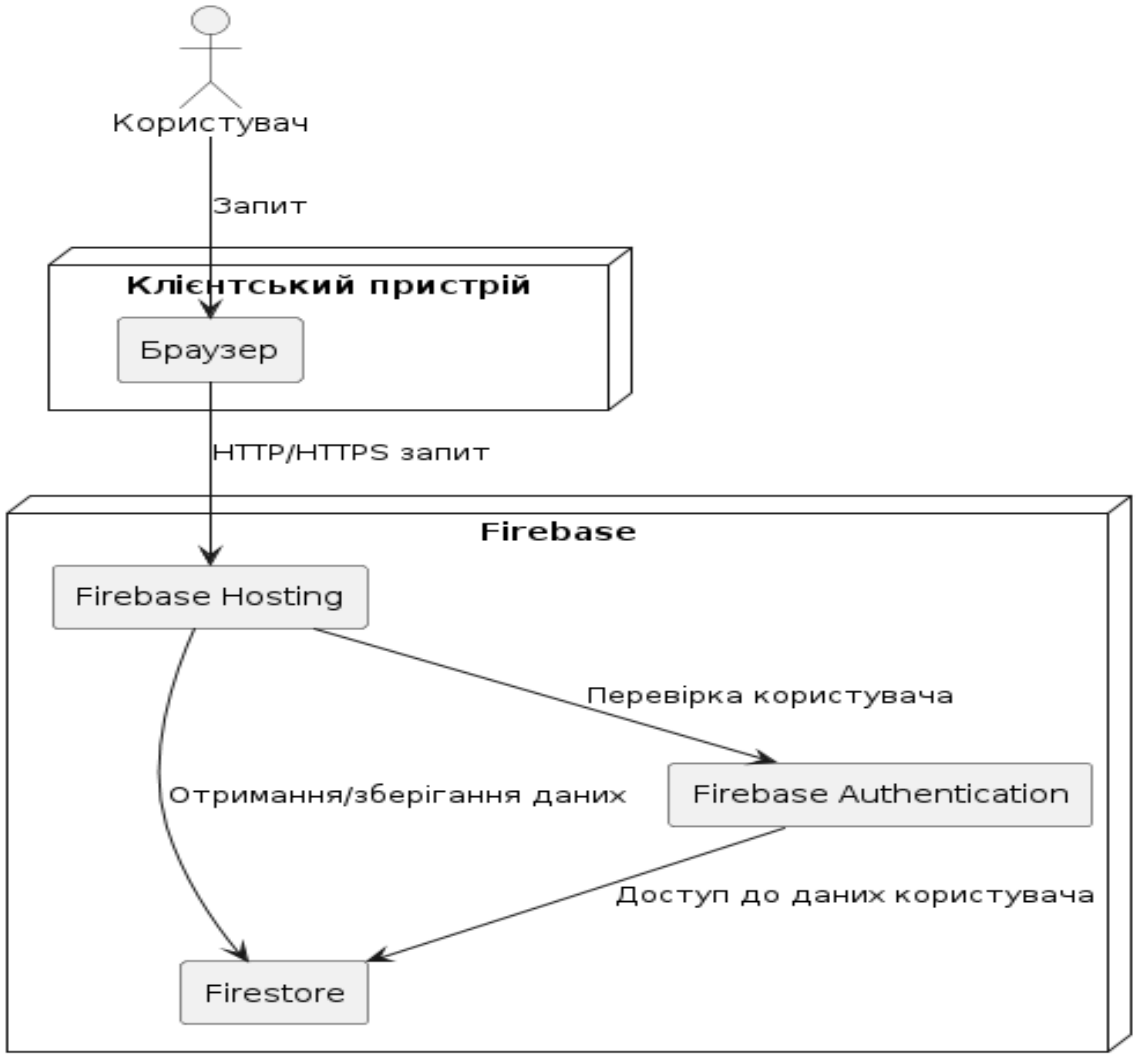
1. Claessens, B. J., van Eerde, W., Rutte, C. G., & Roe, R. A. (2007). A review of the time management literature. Personnel Review, 36(2), 255–276. https://doi.org/10.1108/00483480710726136
2. Dierdorff, E. C. (2020). Time management is about more than life hacks. Harvard Business Review. Retrieved from https://hbr.org/2020/01/time–management–is–about–more–than–life–hacks
3. Häfner, A., Stock, A., Pinneker, L., & Ströhle, S. (2014). Stress prevention through a time management training intervention: An experimental study. Educational Psychology, 34(3), 403–416. https://doi.org/10.1080/01443410.2013.785065
4. König, C. J., & Kleinmann, M. (2005). Time management problems and discounted utility. Journal of Psychology, 139(1), 33–46. https://doi.org/10.3200/JRLP.139.1.33–46
5. Lay, C. H., & Schouwenburg, H. C. (1993). Trait procrastination, time management, and academic behavior. Journal of Social Behavior and Personality, 8(4), 647–662.
6. Macan, T. H. (1994). Time management: Test of a process model. Journal of Applied Psychology, 79(3), 381–391. https://doi.org/10.1037/0021–9010.79.3.381
7. Maddux, J. E., & Tangney, J. P. (Eds.). (2010). Social psychological foundations of clinical psychology. New York, NY: Guilford Press.
8. Mudrack, P. E. (1997). The structure of perceptions of time. Educational and Psychological Measurement, 57(2), 222–240. https://doi.org/10.1177/0013164497057002003
9. Schmitz, B., & Wiese, B. S. (2006). New perspectives for the evaluation of training sessions in self–regulated learning: Time–series analyses of diary data. Contemporary Educational Psychology, 31(1), 64–96. https://doi.org/10.1016/j.cedpsych.2005.02.002

10. Sirois, F. M. (2007). "I'll look after my health, later": An investigation of procrastination and health. Personality and Individual Differences, 43(1), 15–26. https://doi.org/10.1016/j.paid.2006.11.003
11. Steel, P. (2007). The nature of procrastination: A meta–analytic and theoretical review of quintessential self–regulatory failure. Psychological Bulletin, 133(1), 65–94. https://doi.org/10.1037/0033–2909.133.1.65
12. Van Eerde, W. (2003). Procrastination at work and time management training. The Journal of Psychology, 137(5), 421–434. https://doi.org/10.1080/00223980309600625
13. Woolfolk, A. E., & Woolfolk, R. L. (1986). Time management: An experimental investigation. Journal of School Psychology, 24(3), 267–275. https://doi.org/10.1016/0022–4405(86)90047–2
14. Zimmerman, B. J., & Kitsantas, A. (2005). Homework practices and academic achievement: The mediating role of self–efficacy and perceived responsibility beliefs. Contemporary Educational Psychology, 30(4), 397–417. https://doi.org/10.1016/j.cedpsych.2005.05.003
15. Zimbardo, P. G., & Boyd, J. N. (1999). Putting time in perspective: A valid, reliable individual–differences metric. Journal of Personality and Social Psychology, 77(6), 1271–1288. https://doi.org/10.1037/0022–3514.77.6.1271

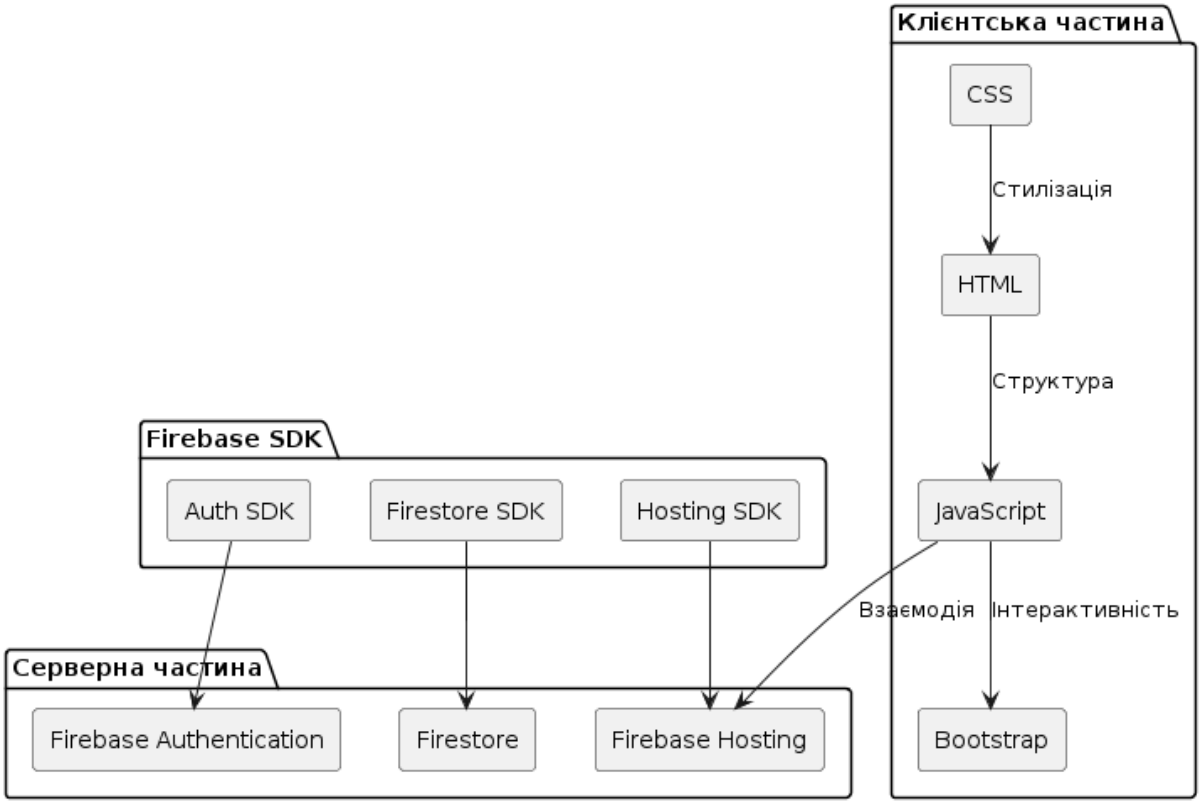
Додатки

Додаток 1

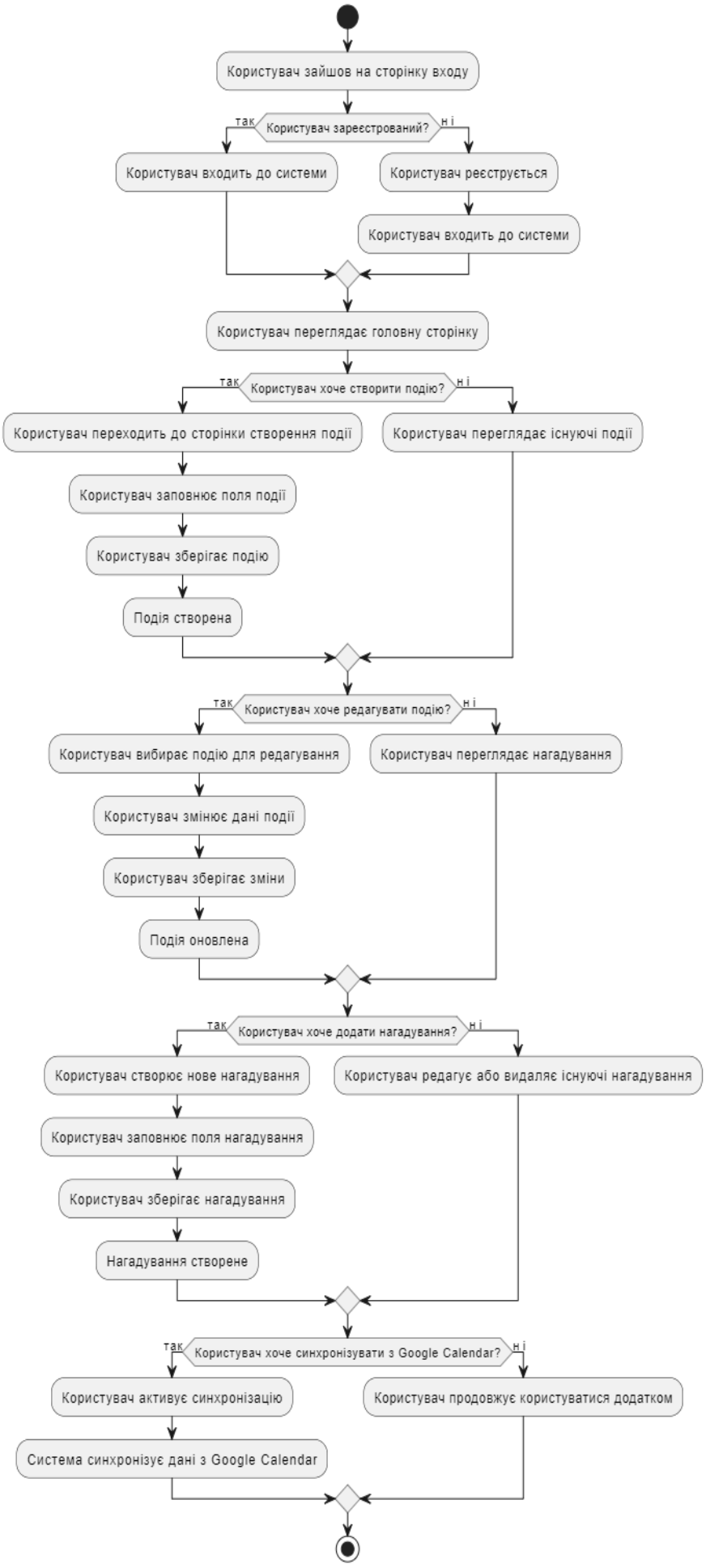
Копії графічних матеріалів



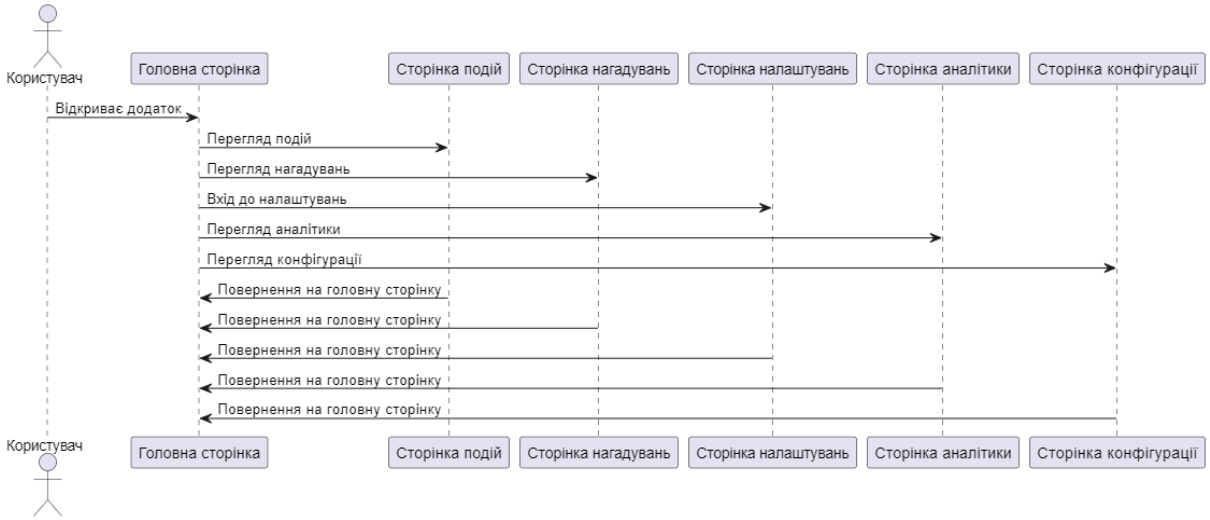
ДП.045440-07-99
Вебзастосунок Schedule manager. Інтеграція між компонентами вебдодатку. UML-діаграма



ДП.045440-08-99
Вебзастосунок Schedule manager.Структура додатку та залежність між компонентами. UML-діаграма



ДП.045440-09-99
Вебзастосунок Schedule manager. Процес авторизації та управління подіями. Діаграма активностей



ДП.045440-10-99
Вебзастосунок Schedule manager. Взаємодія користувача з різними сторінками додатку. Діаграма послідовностей

Додаток 2
Лістинг програми

```
// Auth.css
.auth-container {
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  background-color: #f8f9fa;
}
.auth-form {
  background: #fff;
  padding: 40px;
  border-radius: 8px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
  width: 100%;
  max-width: 400px;
}
.auth-form h2 {
  margin-bottom: 20px;
  font-size: 24px;
  color: #333;
  text-align: center;
}
.auth-form .form-group {
  margin-bottom: 15px;
}
.auth-form .form-group label {
  display: block;
  margin-bottom: 5px;
  color: #333;
}
.auth-form .form-group input {
  width: 100%;
  padding: 10px;
  border: 1px solid #ddd;
  border-radius: 4px;
  font-size: 14px;
}
.auth-form .btn {
  width: 100%;
  padding: 10px;
  border: none;
  border-radius: 4px;
  font-size: 16px;
  cursor: pointer;
}
.auth-form .btn-primary {
  background-color: #007bff;
  color: #fff;
}
.auth-form .btn-primary:hover {
  background-color: #0056b3;
```

```
}
.auth-form .auth-switch {
  margin-top: 20px;
  text-align: center;
}
.auth-form .auth-switch a {
  color: #007bff;
  text-decoration: none;
}

.auth-form .auth-switch a:hover {
  text-decoration: underline;
}
// Home.css
.container {
  max-width: 800px;
  margin: auto;
  padding: 20px;
  background-color: #f8f9fa;
  border-radius: 8px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
.header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  flex-wrap: wrap;
}
.header h1 {
  margin: 0;
  flex: 1 0 100%;
  text-align: center;
}
.header button {
  margin: 10px 0;
  flex: 1 0 auto;
}
.form-inline {
  display: flex;
  justify-content: space-between;
  flex-wrap: wrap;
}
.input-group {
  display: flex;
  align-items: center;
  flex-grow: 1;
  margin-bottom: 10px;
  flex-wrap: wrap;
}
.input-group .form-control {
  margin-bottom: 10px;
```

```
  flex: 1 0 100%;
}
.input-group input[type="date"] {
  flex: 1 0 100%;
  margin-bottom: 10px;
}

.input-group .btn {
  flex: 1 0 100%;
}
.btn-group-wrapper {
  display: flex;
  justify-content: space-between;
  align-items: center;
  flex-wrap: wrap;
  margin-bottom: 10px;
}
.btn-group {
  display: flex;
  flex-direction: column;
  flex: 1 0 100%;
  margin-top: 10px;
}
.btn-group .btn {
  flex: 1 0 100%;
  margin: 5px 0;
}
.alert {
  position: fixed;
  top: 10px;
  right: 10px;
  z-index: 1000;
}
.list-group-item {
  margin-bottom: 10px;
  border-radius: 8px;
  box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
}
.text-decoration-line-through {
  text-decoration: line-through;
}
@media (min-width: 768px) {
  .header h1 {
    flex: 1 0 auto;
    text-align: left;
  }
  .header button {
    flex: 0 0 auto;
    margin: 0;
  }
  .form-inline {
```

```
    flex-direction: row;
  }
  .input-group {
    flex-direction: row;
    flex-wrap: nowrap;
  }
  .input-group .form-control,
  .input-group input[type="date"] {
    flex: 1 0 auto;
    margin-bottom: 0;
    margin-right: 10px;
  }
  .input-group .btn {
    flex: 0 0 auto;
  }
  .btn-group-wrapper {
    flex-direction: row;
    justify-content: space-between;
    align-items: center;
  }
  .btn-group {
    flex-direction: row;
    flex: 0 0 auto;
    width: auto;
    margin-top: 0;
  }
  .btn-group .btn {
    flex: 0 0 auto;
    margin: 0 5px;
  }
}
```

// Home.js

```
import React, { useState, useEffect, useRef } from 'react';
import { useHistory } from 'react-router-dom';
import { db, auth } from '../firebase';
import { collection, addDoc, deleteDoc, doc, updateDoc, onSnapshot }
from 'firebase/firestore';
import { signOut } from 'firebase/auth';
import NotificationSystem from 'react-notification-system';
import 'bootstrap/dist/css/bootstrap.min.css';
import './Home.css';
function Home() {
  const [tasks, setTasks] = useState([]);
  const [newTask, setNewTask] = useState({ title: '', description:
'', dueDate: '', completed: false });
  const [editTask, setEditTask] = useState(null);
  const [filter, setFilter] = useState('all');
  const [searchTerm, setSearchTerm] = useState('');
  const [sort, setSort] = useState('date');
  const history = useHistory();
```

```
  const notificationSystem = useRef(null);
  useEffect(() => {
    const tasksCollection = collection(db, 'tasks');
    const unsubscribe = onSnapshot(tasksCollection, (snapshot) => {
      const tasksList = snapshot.docs.map(doc => ({ id: doc.id,
...doc.data() }));
      setTasks(tasksList);
    });
    return () => unsubscribe();
  }, []);
  useEffect(() => {
    const now = new Date();
    tasks.forEach(task => {
      if (!task.completed && new Date(task.dueDate) <= now) {
        if (notificationSystem.current) {
          notificationSystem.current.addNotification({
            message: `Task "${task.title}" is due!`,
            level: 'warning'
          });
        }
      }
    });
  }, [tasks]);
  const addTask = async (task) => {
    const tasksCollection = collection(db, 'tasks');
    await addDoc(tasksCollection, task);
    if (notificationSystem.current) {
      notificationSystem.current.addNotification({
        message: 'Task added successfully',
        level: 'success'
      });
    }
    setNewTask({ title: '', description: '', dueDate: '', completed:
false });
  };
  const deleteTask = async (id) => {
    await deleteDoc(doc(db, 'tasks', id));
    if (notificationSystem.current) {
      notificationSystem.current.addNotification({
        message: 'Task deleted successfully',
        level: 'success'
      });
    }
  };
  const updateTask = async (id, updatedTask) => {
    await updateDoc(doc(db, 'tasks', id), updatedTask);
    setEditTask(null);
    if (notificationSystem.current) {
      notificationSystem.current.addNotification({
        message: 'Task updated successfully',
        level: 'success'
```

```
      });
    }
  };
  const toggleCompletion = async (id, currentStatus) => {
    await updateDoc(doc(db, 'tasks', id), { completed: !currentStatus
});
    if (notificationSystem.current) {
      notificationSystem.current.addNotification({
        message: `Task marked as ${currentStatus ? 'incomplete' :
'complete'}`,
        level: 'success'
      });
    }
  };
  const handleLogout = async () => {
    await signOut(auth);
    history.push('/login');
  };
  const filteredTasks = tasks.filter(task => {
    if (filter === 'completed') {
      return task.completed;
    } else if (filter === 'incomplete') {
      return !task.completed;
    }
    return true;
  }).filter(task => task.title.includes(searchTerm) ||
task.description.includes(searchTerm));
  const sortedTasks = filteredTasks.sort((a, b) => {
    if (sort === 'date') {
      return new Date(a.dueDate) - new Date(b.dueDate);
    } else if (sort === 'status') {
      return a.completed - b.completed;
    }
    return 0;
  });
  return (
    <div className="container">
      <div className="header my-4 d-flex justify-content-between
align-items-center">
        <h1>My Tasks</h1>
        <button className="btn btn-danger"
onClick={handleLogout}>Logout</button>
      </div>
      <NotificationSystem ref={notificationSystem} />
      <form className="mb-4" onSubmit={(e) => {
        e.preventDefault();
        addTask(newTask);
      }}>
        <div className="input-group mb-3">
```

```
          <input type="text" className="form-control"
placeholder="Title" value={newTask.title} onChange={(e) =>
setNewTask({ ...newTask, title: e.target.value })} required />
          <textarea className="form-control"
placeholder="Description" value={newTask.description} onChange={(e)
=> setNewTask({ ...newTask, description: e.target.value })} required
/>
          <input type="date" className="form-control"
value={newTask.dueDate} onChange={(e) => setNewTask({ ...newTask,
dueDate: e.target.value })} required />
          <button className="btn btn-primary" type="submit">Add
Task</button>
        </div>
      </form>
      <input type="text" className="form-control mb-2"
placeholder="Search tasks..." value={searchTerm} onChange={(e) =>
setSearchTerm(e.target.value)} />
      <div className="btn-group-wrapper d-flex justify-content-
between mb-2">
        <div className="btn-group" role="group">
          <button type="button" className={`btn btn-outline-primary
${filter === 'all' ? 'active' : ''}`} onClick={() =>
setFilter('all')}>All</button>
          <button type="button" className={`btn btn-outline-primary
${filter === 'completed' ? 'active' : ''}`} onClick={() =>
setFilter('completed')}>Completed</button>
          <button type="button" className={`btn btn-outline-primary
${filter === 'incomplete' ? 'active' : ''}`} onClick={() =>
setFilter('incomplete')}>Incomplete</button>
        </div>
        <div className="btn-group" role="group">
          <button type="button" className={`btn btn-outline-secondary
${sort === 'date' ? 'active' : ''}`} onClick={() =>
setSort('date')}>Sort by Date</button>
          <button type="button" className={`btn btn-outline-secondary
${sort === 'status' ? 'active' : ''}`} onClick={() =>
setSort('status')}>Sort by Status</button>
        </div>
      </div>
      <ul className="list-group">
        {sortedTasks.map((task, index) => (
          <li key={index} className="list-group-item d-flex justify-
content-between align-items-center">
            <div>
              <h5 className={task.completed ? 'text-decoration-line-
through' : ''}>{task.title}</h5>
              <p className={task.completed ? 'text-decoration-line-
through' : ''}>{task.description}</p>
              <p className={task.completed ? 'text-decoration-line-
through' : ''}>Due: {task.dueDate}</p>
            </div>
```

```
            <div>
              <button className={`btn ${task.completed ? 'btn-
warning' : 'btn-success'} me-2`} onClick={() =>
toggleCompletion(task.id, task.completed)}>
                {task.completed ? 'Unmark' : 'Complete'}
              </button>
              <button className="btn btn-primary me-2" onClick={() =>
setEditTask(task)}>Edit</button>
              <button className="btn btn-danger" onClick={() =>
deleteTask(task.id)}>Delete</button>
            </div>
          </li>
        ))}
      </ul>
      {editTask && (
        <div className="modal show d-block" tabIndex="-1">
          <div className="modal-dialog">
            <div className="modal-content">
              <div className="modal-header">
                <h5 className="modal-title">Edit Task</h5>
                <button type="button" className="btn-close"
onClick={() => setEditTask(null)}></button>
              </div>
              <div className="modal-body">
                <form onSubmit={(e) => {
                  e.preventDefault();
                  updateTask(editTask.id, editTask);
                  setEditTask(null);
                }}>
                  <div className="mb-3">
                    <label className="form-label">Title</label>
                    <input type="text" className="form-control"
value={editTask.title} onChange={(e) => setEditTask({ ...editTask,
title: e.target.value })} required />
                  </div>
                  <div className="mb-3">
                    <label className="form-label">Description</label>
                    <textarea className="form-control"
value={editTask.description} onChange={(e) => setEditTask({
...editTask, description: e.target.value })} required></textarea>
                  </div>
                  <div className="mb-3">
                    <label className="form-label">Due Date</label>
                    <input type="date" className="form-control"
value={editTask.dueDate} onChange={(e) => setEditTask({ ...editTask,
dueDate: e.target.value })} required />
                  </div>
                  <button type="submit" className="btn btn-
primary">Update Task</button>
                </form>
              </div>
```

```
            </div>
          </div>
        </div>
      )}
    </div>
  );
}

// Login.js
import React, { useState } from 'react';
import { useHistory } from 'react-router-dom';
import { auth } from '../firebase';
import { signInWithEmailAndPassword } from 'firebase/auth';
import './Auth.css';
function Login() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const history = useHistory();
  const handleLogin = async (e) => {
    e.preventDefault();
    try {
      await signInWithEmailAndPassword(auth, email, password);
      history.push('/');
    } catch (error) {
      console.error("Error logging in: ", error);
    }
  };
  return
// START OF App.css
body {
  background-color: #f8f9fa;
  font-family: 'Roboto', sans-serif;
}
.App {
  max-width: 900px;
  margin: auto;
  padding: 20px;
}
h1 {
  color: #343a40;
  margin-bottom: 30px;
}
form {
  margin: 20px 0;
}
input, textarea {
  margin: 5px;
  padding: 10px;
  width: 100%;
}
button {
```

```
  padding: 10px 20px;
  background-color: #007BFF;
  color: white;
  border: none;
  cursor: pointer;
}
button:hover {
  background-color: #0056b3;
}
ul {
  list-style-type: none;
  padding: 0;
}
li {
  background: #ffffff;
  margin: 10px 0;
  padding: 20px;
  border: 1px solid #ddd;
  border-radius: 5px;
}
li h5 {
  margin: 0 0 10px 0;
}
.modal-content {
  border-radius: 10px;
}
.modal-header {
  border-bottom: none;
}
.modal-title {
  font-weight: bold;
}
.btn-close {
  background: transparent;
  border: none;
  font-size: 1.5rem;
}
```

// App.js

```
import React, { useState, useEffect } from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-
dom';
import { onAuthStateChanged } from 'firebase/auth';
import { auth } from './firebase';
import Home from './components/Home';
import Login from './components/Login';
import Register from './components/Register';
import './App.css';

function App() {
  const [user, setUser] = useState(null);
```

```
  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, (user) => {
      setUser(user);
    });
    return () => unsubscribe();
  }, []);
  return (
    <Router>
      <div className="App">
        <Switch>
          <Route path="/" exact component={user ? Home : Login} />
          <Route path="/login" component={Login} />
          <Route path="/register" component={Register} />
        </Switch>
      </div>
    </Router>
  );
}
export default App;

// firebase.js
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";
import { getFirestore } from "firebase/firestore";
const firebaseConfig = {
  apiKey: "AIzaSyAC0NnBHPLzKudxkZdca_-dAX4csKVlEvA",
  authDomain: "scheldue-10a1c.firebaseapp.com",
  projectId: "scheldue-10a1c",
  storageBucket: "scheldue-10a1c.appspot.com",
  messagingSenderId: "506449015378",
  appId: "1:506449015378:web:ee9874f8f15a65c333a58b",
  measurementId: "G-EXHYMHPS31"
};
const app = initializeApp(firebaseConfig);
const db = getFirestore(app);
const auth = getAuth(app);
export { db, auth };

// Home.css
.container {
  max-width: 800px;
  margin: auto;
  padding: 20px;
  background-color: #f8f9fa;
  border-radius: 8px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
.header {
  display: flex;
  justify-content: space-between;
  align-items: center;
```

```
}
.header h1 {
  margin: 0;
}
.header button {
  margin-left: 20px;
}
.form-inline {
  display: flex;
  justify-content: space-between;
  flex-wrap: wrap;
}
.input-group {
  display: flex;
  align-items: center;
  flex-grow: 1;
  margin-bottom: 10px;
}
.input-group .form-control {
  margin-right: 10px;
  flex-grow: 1;
}
.input-group .btn {
  margin-right: 10px;
}
.input-group input[type="date"] {
  flex-basis: 150px;
}
.btn-group-wrapper {
  display: flex;
  justify-content: space-between;
  align-items: center;
  flex-wrap: wrap;
}
.btn-group {
  margin-top: 10px;
  flex-wrap: wrap;
  margin-right: 10px;
}
.alert {
  position: fixed;
  top: 10px;
  right: 10px;
  z-index: 1000;
}
.list-group-item {
  margin-bottom: 10px;
  border-radius: 8px;
  box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
}
.text-decoration-line-through {
```

```
  text-decoration: line-through;
}

// index.js
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import 'bootstrap/dist/css/bootstrap.min.css';
ReactDOM.render(<App />, document.getElementById('root'));
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

ВЕБЗАСТОСУНОК SHEDULE MANAGER

Виконав: Шилов Едуард Юрійович

Керівник: доцент кафедри ПЗКС, к.т.н. доцент,
Новак Дмитро Сергійович

Київ – 2024

ПОСТАНОВКА ЗАДАЧІ

Мета проекту: створення зручного веб-інструменту для ефективного планування, організації та моніторингу виконання різноманітних завдань і подій в єдиному календарному інтерфейсі.

Завдання:

1. Проаналізувати ринок вебзастосунків
2. Порівняти аналоги з розроблюваним вебзастосунком
3. Розробити перелік вимог для вебзастосунку
4. Обрати технологічні рішення для розробки застосунку
5. Розробити вебзастосунок згідно переліку вимог та обраних технологій
6. Провести тестування готового продукту

АКТУАЛЬНІСТЬ

- Згідно статистичних досліджень, попит на інструменти для управління часом зростає на 20% щороку.
- Люди, які використовують подібні інструменти, збільшують свою продуктивність на 30%.

АНАЛОГІЧНІ РІШЕННЯ

Todoist

Trello

Asana

ФУНКЦІОНАЛЬНІСТЬ РОЗРОБЛЮВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Модуль замовника та виконавця

1. Створення/редагування/видалення подій та завдань з пріоритетами та термінами
2. Підтримка різних типів подій (одноразові, циклічні та інші)
3. Категоризація та фільтрація завдань
4. Перегляд розкладу в різних режимах (день, тиждень, місяць)
5. Синхронізація з Google Calendar, Exchange тощо
6. Адаптивний та зручний користувацький інтерфейс
7. Система сповіщень для нагадувань про наближення термінів

ФУНКЦІОНАЛЬНІСТЬ РОЗРОБЛЮВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

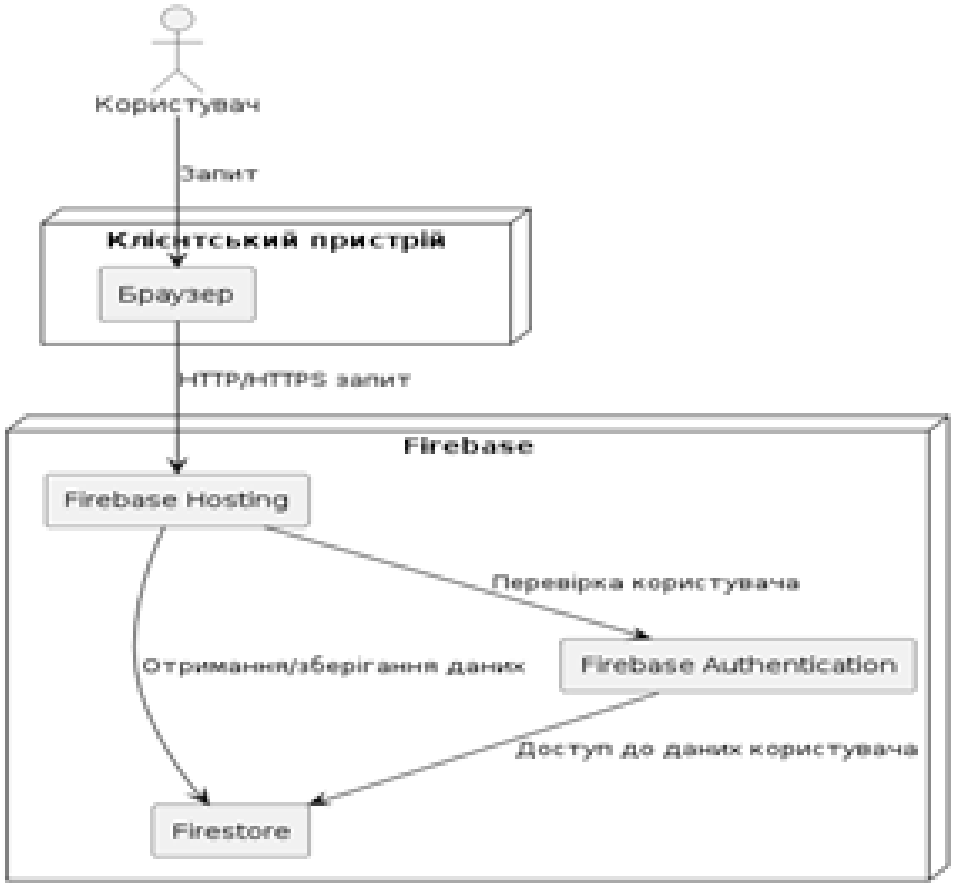
Модуль адміністратора

1. Видаляти, редагувати публікації інших користувачів
2. Переглядати, редагувати, видаляти аккаунти замовників та виконавців
3. Можливість переглядати всі існуючі завдання
4. Можливість змінювати права доступу виконавцям

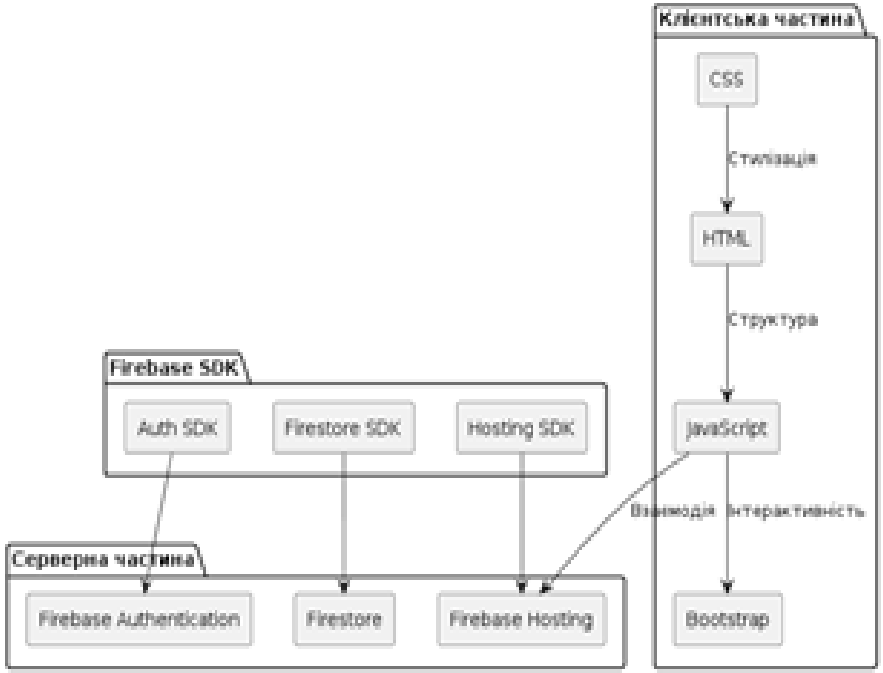
ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

Фронтенд: HTML, CSS, JavaScript
Бекенд: Firebase
Аутентифікація: Firebase Authentication
База даних: Firestore, NoSql

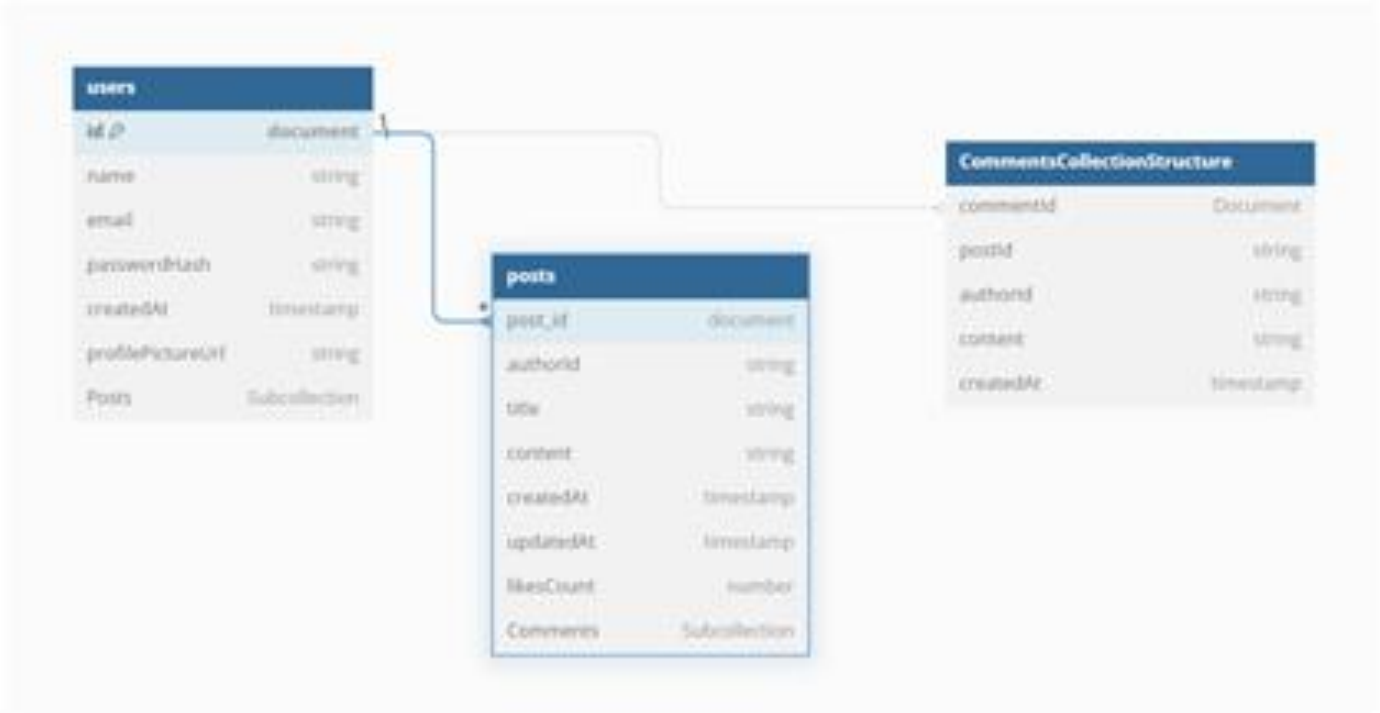
АРХІТЕКТУРА ЗАСТОСУНКУ



АРХІТЕКТУРА ЗАСТОСУНКУ



СТРУКТУРА БД

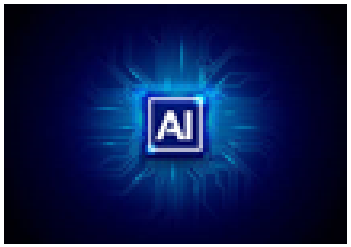


ПОРІВНЯННЯ З АНАЛОГАМИ

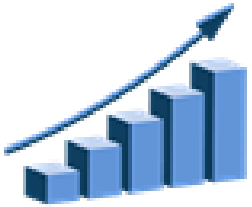
Під час порівняння вебзастосуноку з іншими конкурентами виявлено, що додаток має наступні переваги:

1) Повна гнучкість у додаванні нових функцій
2) Спрощений процес введення даних та персоналізовані налаштування
3) Висока безпека даних
4) Зручний інтуїтивно зрозумілий інтерфейс
5) інтеграцію з внутрішніми системами, гнучкість та масштабованість

ШЛЯХИ ПОДАЛЬШОГО РОЗВИТКУ



Підключення AI



Рейтинг користувачів

Оптимізація інтерфейсу

ВИСНОВКИ

1. Проведено аналіз ринку аналогічних вебзастосунків.
2. Проведено порівняння засобів розробки та обрані оптимальні.
3. Описано загальну структуру та архітектуру програмного застосунку.
4. Розроблено вебзастосунок згідно переліку вимог та обраних технологій
5. Сформовано рекомендації для подальшого розвитку та підтримки вебзастосунку.

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2023 р.

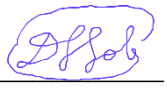
ВЕБЗАСТОСУНОК SCHEDULE MANAGER

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проєкту:

 Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Едуард ШИЛОВ

2024

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Об'єктом випробувань є вебдодаток "Schedule Manager" – система управління часом та організації робочого процесу, розроблена із застосуванням сучасних веб-технологій.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) функціональна працездатність елементів сторінок вебзастосунку;
2) забезпечення належного рівня безпеки даних;
3) зручність роботи з веб-сайтом;
4) відповідність дизайну вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

1) функціональне тестування, зокрема на рівні Critical path test (базове тестування);
2) тестування продуктивності програмного забезпечення, зокрема Stability testing (тестування стабільності) та Load testing (навантажувальне тестування);
3) тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується засобами інструментарію SpecFlow.

Працездатність вебзастосунку перевіряється шляхом:

1) динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;

2) динамічного ручного тестування на відповідність функціональним вимогам;
3) статичного тестування коду;
4) тестування вебзастосунку в різних веб-браузерах;
5) тестування при максимальному навантаженні;
6) тестування стабільності роботи при різних умовах;
7) тестування зручності використання;
8) тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2024 р.

ВЕБЗАСТОСУНОК SCHEDULE MANAGER

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проєкту:

 Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Едуард ШИЛОВ

2024

ЗМІСТ

1.Опис структури вебзастосунку "Schedule Manager"

Вебзастосунок "Schedule Manager" призначений для організації та управління розкладом користувачів. Його структура включає як статичні, так і динамічні сторінки.

1. Статичні вебсторінки

До статичних сторінок належать:

- «Про додаток» / «About the App»: Інформація про вебзастосунок.
- «Функціональні можливості» / «Features»: Опис основних функцій.
- «Інструкції з використання» / «User Guide»: Інструкції для користувачів.
- «Контакти» / «Contacts»: Контактна інформація.

2. Динамічні вебсторінки

Динамічні сторінки забезпечують інтерактивність:

- Персональна сторінка користувача: Інформація про користувача, його завдання та налаштування.
- Сторінка всіх завдань: Список завдань і подій, доступних користувачу.
- Сторінка виконаних завдань: Список завдань і подій, виконаних
- користувачем.
- Сторінка невиконаних завдань: Список завдань і подій, невиконаних користувачем.

3. Адміністративна панель

Вебзастосунок також має адміністративну панель, доступну лише адміністратору ресурсу. Панель містить інструменти для управління користувачами, завданнями та налаштуваннями системи.

4. Загальна структура

- Клієнтська частина (фронтенд): HTML, CSS, JavaScript.

- Серверна частина (бекенд): Firebase, Firebase Hosting, Firebase Authentication для аутентифікації користувачів.
- База даних: Firestore, яка є NoSQL базою даних.

2. Опис вмісту загальної вебсторінки

На основній веб–сторінці "Schedule Manager" відображаються основні функціональні елементи, що дозволяють користувачеві легко управляти своїм розкладом:

- Заголовок "My Tasks": Вказує на основний зміст сторінки (перелік завдань користувача).
- Форма для додавання завдань: Містить поля для введення назви завдання (Title), його опису (Description) та дати (ДД.ММ.РРРР). Кнопка "Add Task" дозволяє додати нове завдання до списку.
- Пошуковий рядок: Дозволяє користувачеві швидко знаходити завдання за ключовими словами.
- Фільтри завдань: Кнопки "All", "Completed" та "Incomplete" дозволяють фільтрувати завдання за їх статусом (усі, завершені, незавершені).
- Сортування завдань: Кнопки "Sort by Date" та "Sort by Status" дозволяють сортувати завдання за датою або статусом.
- Кнопка виходу "Logout": Дозволяє користувачеві вийти з облікового запису.

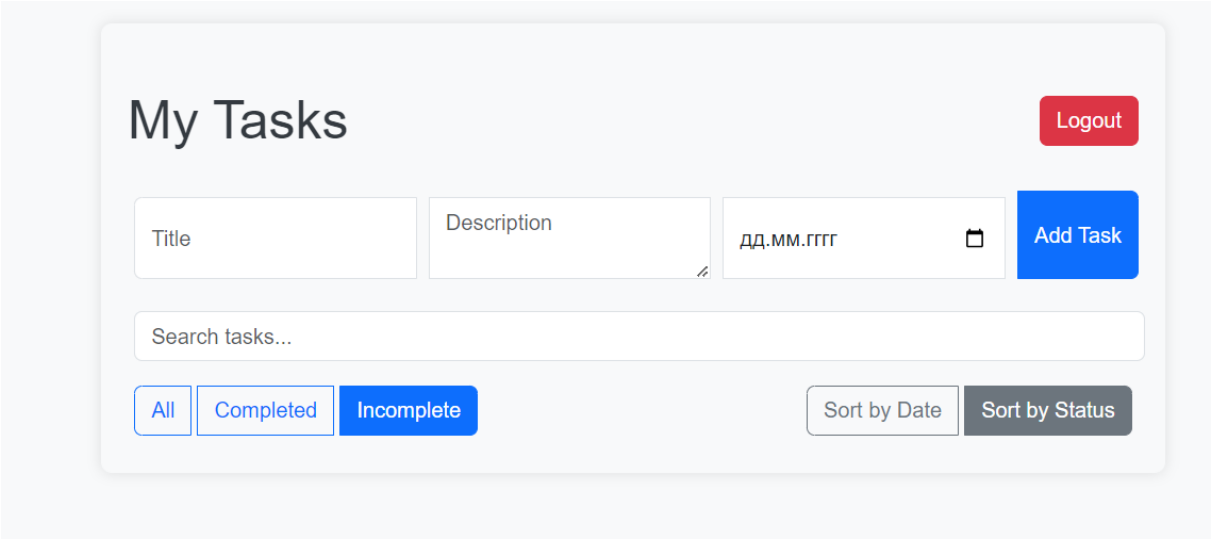


Рис. 2.1. Основна вебсторінка

Всі сторінки містять посилання на головну сторінку ресурсу, посилання для переходу до основних розділів та додаткові функціональні елементи для зручного використання.

3. Процедура авторизації та реєстрації користувача

Для реєстрації в застосунку вгорі є заголовок "Register" (Реєструватися). Нижче є поле вводу для введення електронної адреси користувача. В даному випадку, в поле вводу вже введено надану для прикладу адресу "shylov.eduard@gmail.com".Під полем вводу електронної адреси є поле для введення пароля, пароль приховано за символами "•••••••••".

Внизу форми є кнопка "Register" (Реєструватися) синього кольору для завершення процесу реєстрації.

Також є посилання "Already have an account? Login here" (Вже маєте акаунт? Увійдіть тут), яке дозволяє перейти до сторінки входу для існуючих користувачів.

Загалом, ця сторінка містить типові елементи форми реєстрації, такі як поля вводу електронної адреси та пароля, а також кнопку для підтвердження та посилання на сторінку входу.

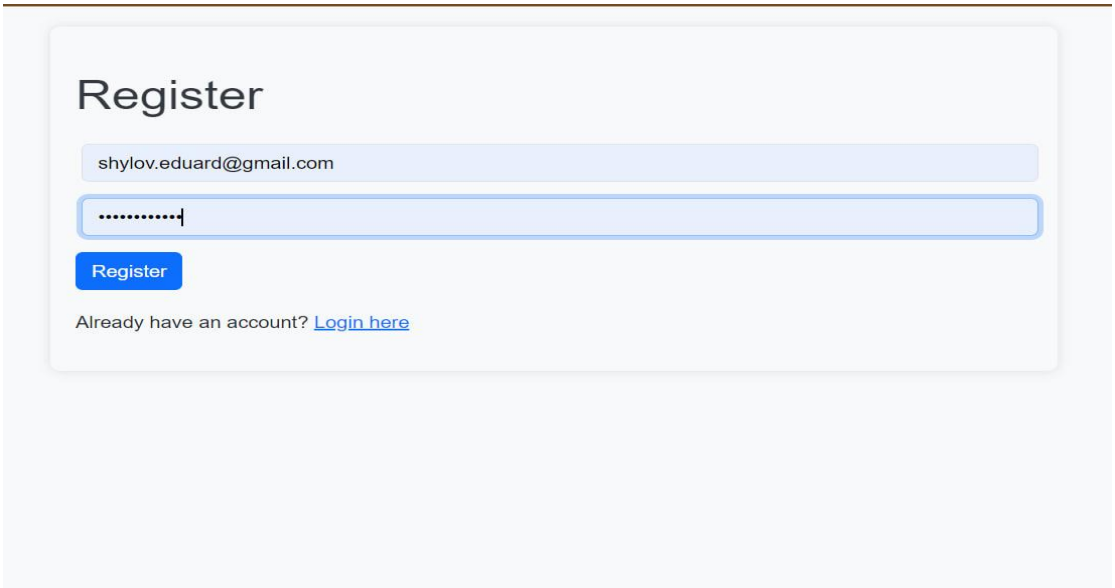


Рис. 2.2. Форма реєстрації

Авторизація користувача відбувається на сторінці «Login». Користувач має ввести свої логін і пароль та натиснути кнопку «Login», після чого відбувається перехід до головної сторінки. Нижче наведено приклад сторінки входу:

1. Після успішної авторизації користувач отримує доступ до всіх функцій системи.
2. У разі невірного введення даних відображається повідомлення про помилку.
3. Для відновлення пароля є спеціальна опція "Забули пароль?".