НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Програмне забезпечення для управління та моніторингу роботи коворкінгів

Виконав студент IV курсу, групи ІТ-92
(шифр групи)

Русановський Дмитро Дмитрович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник ст. викладач, Вітковська І. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації ст. викладач, Вітковська І. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент проф., д.т.н., проф. каф. ОТ, Писарчук О. О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Русановському Дмитру Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Програмне забезпечення для управління та моніторингу роботи коворкінгів

керівник проєкту Вітковська І. І., ст. викладач
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « 31 »травня 2023 р. № 2101-с

2. Термін подання студентом проєкту « 17 » червня 2023 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення.
2) Моделювання та конструювання програмного забезпечення.
3) Аналіз якості та тестування програмного забезпечення.
4) Впровадження та супровід програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема бази даних

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» березня 2023 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2023	
2	Аналіз існуючих методів розв’язання задачі	17.03.2023	
3	Постановка та формалізація задачі	17.03.2023	
4	Алгоритмізація задачі	31.03.2023	
5	Проектування структури програмного забезпечення	05.04.2023	
6	Обґрунтування вибору використаних технічних засобів	05.04.2023	
7	Розробка програмного забезпечення	07.05.2023	
8	Налагодження програми	14.05.2023	
9	Виконання графічних документів	22.05.2023	
10	Оформлення пояснювальної записки	22.05.2023	
11	Подання ДП на попередній захист	23.05.2023	
12	Подання ДП рецензенту	12.06.2023	
13	Подання ДП на основний захист	17.06.2023	

Студент _____ Дмитро РУСАНОВСЬКИЙ
(підпис) (ініціали, прізвище)

Керівник _____ Ірина ВІТКОВСЬКА
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 30 таблиць, 18 рисунків та 10 джерел – загалом 80 сторінки.

Дипломний проєкт присвячений розробці програмного забезпечення для управління та моніторингу роботи коворкінгів.

Мета даного проекту є підвищення ефективності організації робочих процесів та зниження трудомісткості управління ресурсами коворкінгів. Метою розробки є створення універсального інструменту, що полегшить управління коворкінгами, забезпечить прозорість використання простору, автоматизує адміністративні процеси та сприятиме ефективності використання ресурсів. Розробка спрямована на забезпечення зручного досвіду користування для керівників коворкінгів, їх персоналу, а також відвідувачів та користувачів цих просторів.

Об'єкт дослідження: Програмне забезпечення управління та моніторингу роботи коворкінгів.

Предмет дослідження: Процеси автоматизації та оптимізації роботи коворкінгів за допомогою програмного забезпечення, а також аналіз основних викликів та проблем, пов'язаних з їхнім управлінням та моніторингом.

У розділі першому досліджена предметна область, розглянуті технічні рішення і аналоги, розроблені вимоги.

Розділ другий присвячений моделюванню бізнес-процесів, створенню архітектури і конструюванню програмного забезпечення.

У третьому розділі розглядається аналіз якості програмного забезпечення та описуються процеси тестування.

У четвертому розділі розглядається розгортання і випуск програмного забезпечення.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ДОДАТОК, PROGRESSIVE WEB APPLICATION, ANDROID, GOOGLE FIREBASE, БАЗА ДАНИХ.

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 30 tables, 18 figures and 10 sources – in total 80 pages.

The diploma project is dedicated to the development of software for managing and monitoring of co-working spaces.

The purpose of development is to create a universal tool that will simplify the management of coworking spaces, ensure transparency of space usage, automate administrative processes, and promote the effective use of resources. The development aims to provide a convenient user experience for coworking managers, their staff, as well as visitors and users of these spaces.

Object of study: Software for managing and monitoring the operation of coworking spaces.

Subject of study: The processes of automation and optimization of coworking operation using software, as well as the analysis of main challenges and problems related to their management and monitoring.

The first section investigates the subject area, considers technical solutions and analogs, develops requirements.

The second section is dedicated to business process modeling, creating architecture, and constructing software.

The third section discusses the software quality analysis and describes testing processes.

The fourth section examines software deployment and release.

KEYWORDS: MOBILE APPLICATION, PROGRESSIVE WEB APPLICATION, ANDROID, GOOGLE FIREBASE, DATABASE.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-9221.045440.01.91	Технічне завдання	17	
3	А4	КПІ.ІТ-9221.045440.02.81	Пояснювальна записка	80	
4	А4	КПІ.ІТ-9221.045440.03.12	Текст програми	88	
5	А4	КПІ.ІТ-9221.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІТ-9221.045440.05.34	Керівництво користувача	15	
7	А3	КПІ.ІТ-9221.045440.06.99	Графічні матеріали	3	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-9121.045440.00.90		
Розроб.		Русановський Д.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Вітковська І. І.					
Н.контр.		Вітковська І. І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-92	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2023 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ ТА МОНІТОРИНГУ РОБОТИ КОВОРКІНГІВ

Технічне завдання

КПІ.ІТ-9221.045440.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Ірина ВІТКОВСЬКА

Нормоконтроль:

_____ Ірина ВІТКОВСЬКА

Виконавець:

_____ Дмитро РУСАНОВСЬКИЙ

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: програмне забезпечення для управління та моніторингу роботи коворкінгів.

Галузь застосування: розробка системи управління, орієнтованої на сектор нерухомості та гостьового господарства, зокрема на управління просторами спільного використання — коворкінги. Програмне забезпечення також може бути корисним для розробників нерухомості та операторів просторів спільного користування, які шукають ефективні технології для управління та оптимізації своїх ресурсів. Воно також може слугувати як інструмент для муніципалітетів та організацій громадського сектора, які прагнуть створити та ефективно управляти ком'юніті-центрами та іншими просторами спільного користування. Таким чином, програмне забезпечення має потенціал для використання в широкому спектрі галузей, включаючи сектор нерухомості, гостьове господарство, освіту, стартап-екосистему, громадський сектор та багато інших.

Наведене технічне завдання поширюється на розробку програмного забезпечення для управління та моніторингу роботи коворкінгів [КПІ.ІТ-9221.045440.02.81], котра використовується для автоматизації та оптимізації процесів управління та моніторингу в коворкінгах. Це включає у себе управління бронюваннями та резервуваннями приміщень, відстеження використання ресурсів, контролю дотримання правил користування, а також аналізу даних для прийняття бізнес-рішень. та призначена для широкого спектру користувачів, включаючи власників коворкінгів, менеджерів приміщень, співробітників служби підтримки, а також клієнтів, які використовують коворкінги. Воно може бути корисним у сфері коворкінгів та просторів спільного користування в містах і регіонах, де є високий попит на такі послуги.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки програмного забезпечення для управління та моніторингу роботи коворкінгів є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для автоматизації процесів управління та моніторингу роботи коворкінгів, що передбачає планування робочого простору, контроль доступу, резервування місць, відстеження використання ресурсів, бронювання конференційних залів та інші важливі функції.

Метою розробки є створення універсального інструменту, що полегшить управління коворкінгами, забезпечить прозорість використання простору, автоматизує адміністративні процеси та сприятиме ефективності використання ресурсів. Розробка спрямована на забезпечення зручного досвіду користування для керівників коворкінгів, їх персоналу, а також відвідувачів та користувачів цих просторів. Розробка покликана значно спрощувати управління просторами, сприяти їх легкій адаптації до змінюваних умов роботи, різних потреб користувачів та викликів ринку.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– адаптивність під різні типи мобільних пристроїв;

– інтерфейс повинен бути інтуїтивно зрозумілим і легким у використанні. Він повинен містити всі необхідні елементи управління для доступу до основних функцій програми;

– має бути наявний механізм авторизації/реєстрації, що включає в себе поля для введення електронної пошти та паролю, а також кнопки для підтвердження введених даних (рисунок 1.1);

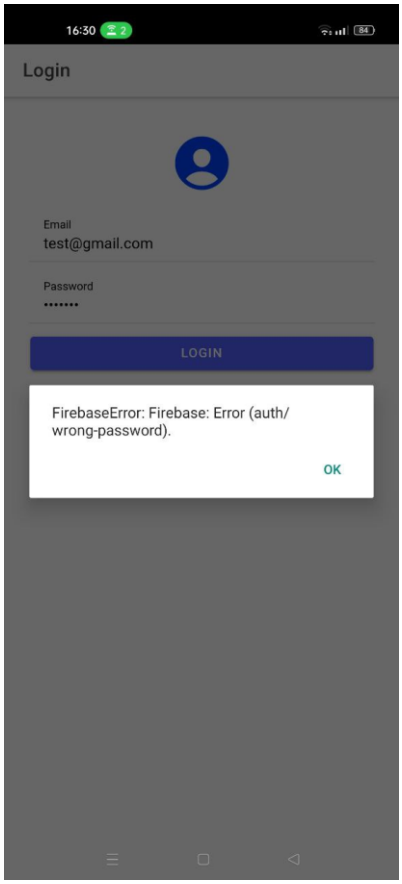


Рисунок 1.1 — прототип сторінки реєстрації та авторизації

– функція пошуку та перегляду доступних для бронювання коворкінгів повинна включати в себе динамічний список коворкінгів, а також фільтри для вибору специфічних параметрів (рисунок 1.2);

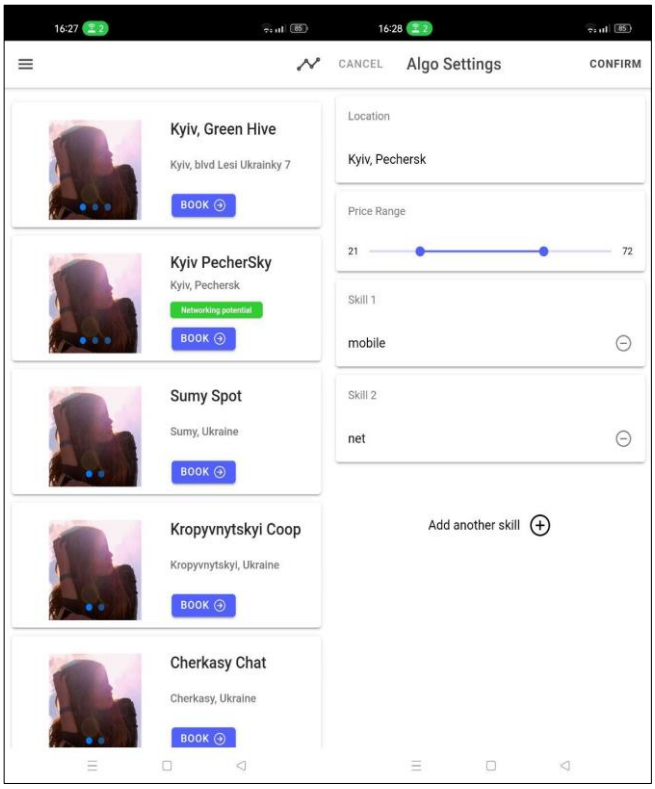


Рисунок 1.2 — прототип сторінки пошуку та фільтру коворкінгів

– функція створення бронювання повинна включати календар для вибору дати та часу, а також динамічний список доступних робочих місць в обраному коворкінгу (рисунок 1.3);

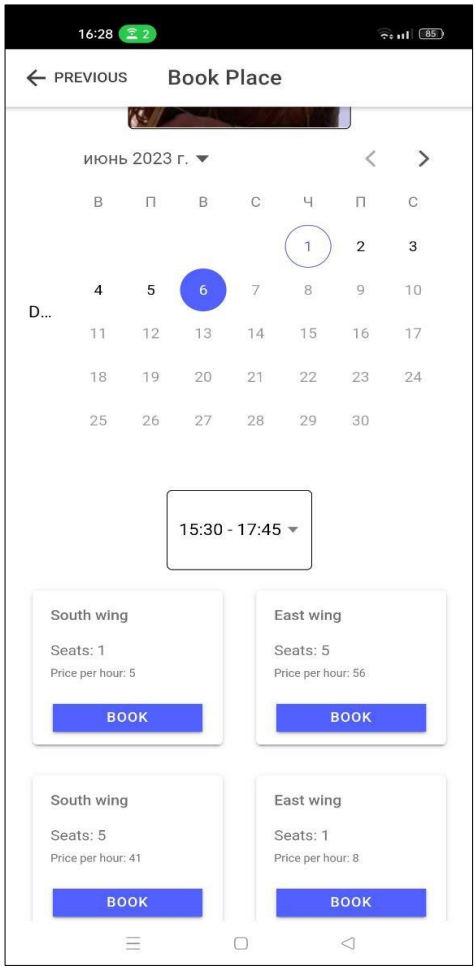


Рисунок 1.3 — прототип сторінки бронювання робочих місць

– перегляд усіх бронювань та зміна статусу бронювань користувача повинні бути організовані в форматі списку, де кожне бронювання представлене в окремому елементі (рисунок 1.4);

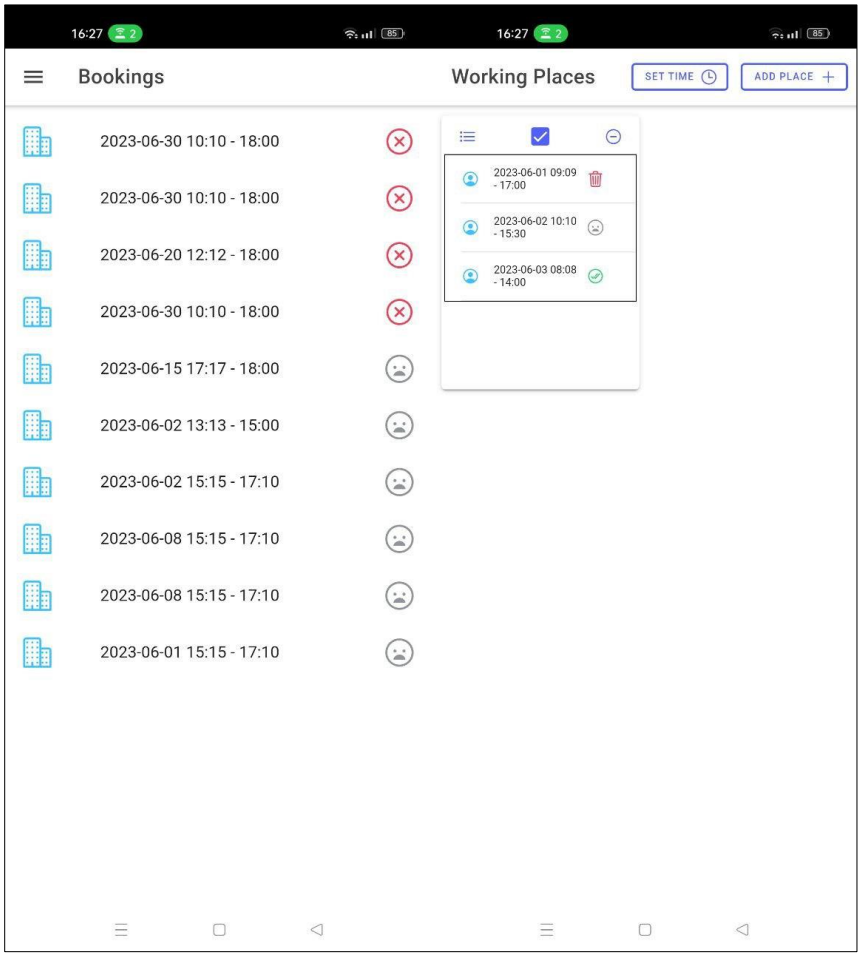


Рисунок 1.4 — прототип сторінок бронювання робочих місць користувача та адміністратора коворкінгу

– для адміністратора коворкінгу повинна бути доступна функція управління інформацією про коворкінг, включаючи реєстрацію нового коворкінгу, додавання/видалення робочих місць, задавання доступних дат та часів для бронювання (рисунок 1.5);

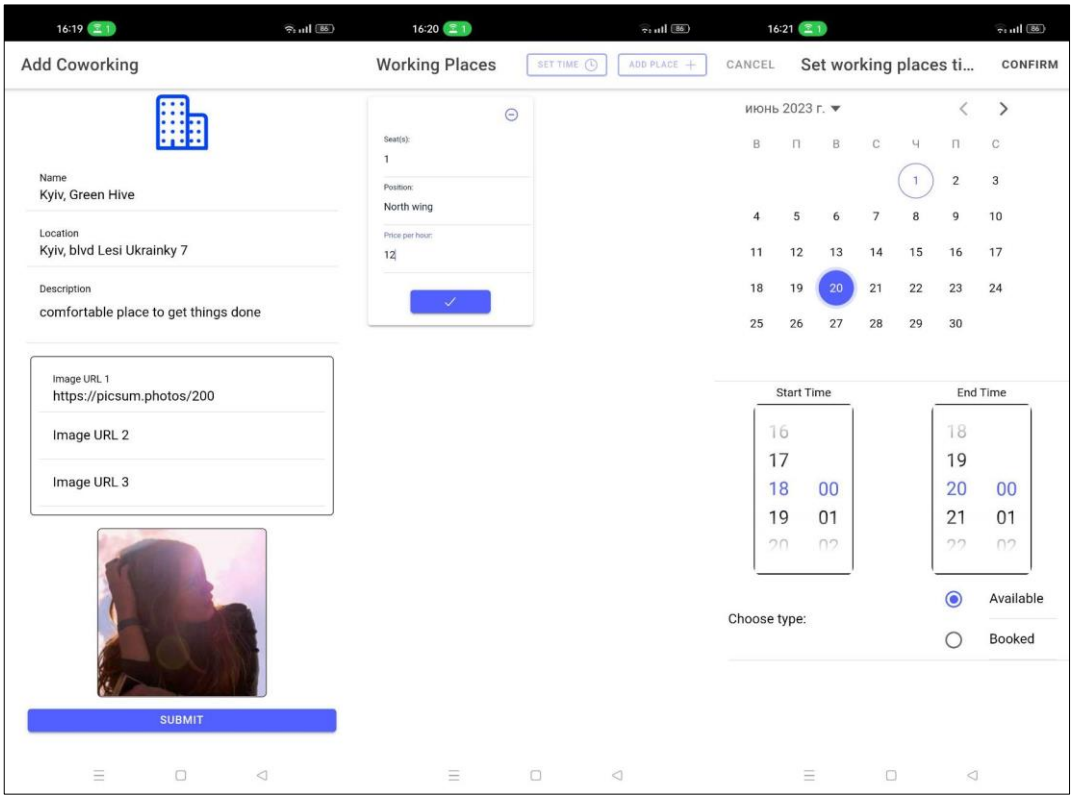


Рисунок 1.5 — прототип сторінок реєстрації коворкінгу, додавання робочих місць, задавання доступних часових слотів

4.1.2 Для користувача:

– реєстрація користувача за допомогою пошти і пароля;
– авторизація користувача за допомогою пошти і паролю;
– пошук та перегляд доступних для бронювання коворкінгів;
– зміна пошукових параметрів для коворкінгів та навичок користувача;
– створення бронювання на задану дату, час і робоче місце в обраному коворкінгу;
– перегляд усіх бронювань та зміна статусу бронювань користувача.

4.1.3 Для адміністратора коворкінгу:

– реєстрація адміністратора за допомогою пошти і пароля;
– авторизація адміністратора за допомогою пошти і паролю;
– реєстрація та додавання інформації по своєму коворкінгу;
– додавання та видалення робочих місць в коворкінгу;

– задавання доступної дати та часу для бронювання робочим місцям;

– додавання зовнішніх бронювань робочим місцям для відображення в системі;

– перегляд заданих доступних дати і часу для бронювання робочих місць;

– перегляд усіх бронювань робочих місць та зміна статусів активних бронювань.

4.1.4 Додаткові вимоги:

– оновлення статусів активних бронювань, якщо їх час сплив.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних. В разі виникнення помилок або збоїв, система повинна надавати зрозумілі повідомлення про проблеми і способи їх вирішення. Повинен бути реалізований механізм збереження цілісності даних в БД при виконанні операцій модифікації.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Розробка не потребує спеціалізованого обслуговування. Проте, для підтримки надійної та безперебійної роботи програмного забезпечення рекомендується регулярне оновлення системи до останніх версій.

Також, може бути необхідною технічна підтримка користувачів при виникненні проблем з використанням програмного забезпечення. Для цього

може бути організований сервіс підтримки, який буде надавати допомогу користувачам при виникненні питань або проблем.

Крім того, регулярно необхідно проводити аналіз безпеки системи та виконувати необхідні оновлення для захисту від потенційних загроз.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Розроблене програмне забезпечення призначене для використання на мобільних пристроях з операційною системою Android або iOS, а також на комп'ютерах за допомогою веб-браузера.

Мінімальна конфігурація технічних засобів для мобільних пристроїв:

– операційна система: Android 8.0, iOS 11.0 або новіші версії;
– об'єм оперативної пам'яті: 2 Гб;
– мінімальний розмір екрану: 4.7 дюймів;
– підключення до мережі Інтернет зі швидкістю від 10 мегабіт/сек.

Мінімальна конфігурація технічних засобів для веб-версії:

– операційна система: Windows 7, MacOS 10.12 Sierra, Linux або новіші версії;
– веб-браузер: Google Chrome 60, Firefox 55, Safari 10, Microsoft Edge 14 або новіші версії;
– об'єм оперативної пам'яті: 2 Гб;
– підключення до мережі Інтернет зі швидкістю від 10 мегабіт/сек.

Рекомендована конфігурація технічних засобів:

– для мобільних пристроїв — Android 10.0, iOS 13.0 або новіші версії, 4 Гб ОЗП, підключення до мережі Інтернет зі швидкістю не менше 20 мегабіт/сек;

– для веб-версії — персональний комп'ютер або ноутбук з операційною системою Windows 10, MacOS 10.15 Catalina, Linux або новіші версії, 4 Гб ОЗП, веб-браузер Google Chrome 80 або новіші версії, підключення до мережі Інтернет зі швидкістю не менше 20 мегабіт/сек.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно бути сумісне з веб-браузерами Google Chrome, Mozilla Firefox, Safari, Microsoft Edge, та мобільними операційними системами Android та iOS.

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в форматах, які сумісні з JSON (для обміну даними між сервером і клієнтом).

4.5.2 Вимоги до вихідних даних

Вихідні дані повинні бути представлені в форматі JSON для подальшого відображення на клієнтській стороні.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Typescript (для клієнтської частини) і Node.js Javascript (для серверної частини).

4.5.4 Вимоги до середовища розробки

Розробку веб-версії (клієнтська і серверна частини) виконати в середовищі розробки Visual Studio Code. Для розробки Android-версії використовувати середовище Android Studio.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код клієнтської частини має бути представлений у вигляді текстових файлів з розширенням .ts та .js для серверної частини, організованих в відповідних папках проекту.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати інсталяційну версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– схема бази даних;
– архітектура програмного забезпечення;
– креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.05	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

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

Пояснювальна записка
до дипломного проєкту

на тему: “Програмне забезпечення для управління та моніторингу роботи коворкінгів”

КПІ.ІТ-9221.045440.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE – Integrated Development Environment – інтегроване середовище розробки.

API – Application programming interface, прикладний програмний Інтерфейс

SDK – Software development kit

IT – Інформаційні технології

ER – Entity-Relation diagram

ОС – Операційна система.

БД – База даних.

PWA – Progressive Web App

ВСТУП

Сучасний світ швидко змінюється, і це стосується всіх сфер нашого життя. Однією з ключових тенденцій, що формують наше суспільство, є перехід від традиційних форм праці до більш гнучких та рухливих. Коворкінги, як місця для спільної роботи та обміну ідеями, стають все більш популярними, оскільки вони відповідають цій тенденції, надаючи простір для інновацій та співпраці.

Відповідно до зростаючого попиту і важливості коворкінгів, стає актуальною розробка ефективного програмного забезпечення, що сприяє управлінню та моніторингу роботи коворкінгів. На сьогодні існує багато програмних рішень, проте вони часто зосереджені на основних операційних потребах та не враховують специфіку можливого нетворкінгу. Адже саме нетворкінг - взаємодія та обмін знаннями між учасниками - є одним із ключових аспектів, що приваблюють користувачів до коворкінгів.

Сфера застосування програмного забезпечення для управління та моніторингу роботи коворкінгів значно перевищує межі простого управління просторами. Це не обмежується тільки провайдерами коворкінгів, але і включає в себе широкий спектр зацікавлених сторін: організації та індивідуалів, що використовують коворкінги як місце роботи; великі корпорації, що хочуть розуміти динаміку використання робочого простору і включати новітні технології в свою роботу; і навіть урядові організації, що мають за ціль підтримувати інноваційні простори в своїх містах для стимулювання економічного зростання і розвитку технологій.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Коворкінги - це сучасні простори для спільної роботи, які стали популярними серед фрілансерів, стартапів та невеликих компаній. Вони пропонують гнучкі варіанти праці, у порівнянні з традиційним орендуванням офісного простору [1]. Процес управління коворкінгами, включаючи резервування робочих місць, організацію заходів, управління відгуками клієнтів і т.д., часто вимагає ефективного програмного забезпечення.

Коворкінги активно розвиваються на фоні стійкого росту гіг-економіки і тенденції до дистанційної роботи, що прогнозується лише посилитися в майбутньому [3]. За допомогою коворкінгів, фрілансери, незалежні професіонали та невеликі команди здатні знайти доступне робоче місце, що включає всі необхідні ресурси, а також можливість взаємодії з іншими професіоналами.

Понад це, коворкінги забезпечують значні можливості для нетворкінгу. Міждисциплінарність коворкінгів стимулює обмін ідеями та колаборацію між різними професійними групами, що може сприяти інноваційному розвитку [4].

Застосунок для керування та моніторингу роботи коворкінгів може спростити процес пошуку та резервування робочих місць, а також організації та відвідування заходів у коворкінгу для користувачів. Він також допомагає власникам коворкінгів керувати ресурсами, контролювати резервування та отримувати відгуки від користувачів. Додатково, за допомогою використання інтелектуального алгоритмічного підбору коворкінгів, застосунок може підвищити ефективність нетворкінгу користувачів, надаючи результати, що найбільше відповідають їхнім професійним потребам.

Розробка програмного забезпечення для управління та моніторингу роботи коворкінгів включає створення рішення, що дозволяє власникам коворкінгів керувати їхніми ресурсами і слідкувати за використанням, а

користувачам - легко знайти та забронювати місця в коворкінгах [2]. Подібні системи можуть бути реалізовані через веб- та мобільні застосунки, крос-платформено. Можуть включати алгоритми рекомендацій, інтеграцію з соціальними мережами, системи оцінювання та відгуків, системи управління бронюваннями, а також системи управління заходами.

Кросплатформеність є ключовим аспектом у створенні сучасних додатків, оскільки вона забезпечує доступ до додатку для користувачів різних платформ. Це особливо важливо для таких застосунків, як наш додаток для коворкінгів, що призначений для широкого кола користувачів, включаючи фрілансерів, власників малого бізнесу, незалежних професіоналів і т.д.

1.2 Змістовний опис і аналіз предметної області

Коворкінги стали важливою складовою сучасного бізнес-середовища, надаючи місце для роботи і нетворкінгу професіоналів з різних сфер. Однак, реалізація цього потенціалу у програмному забезпеченні залишається недостатньо розробленою.

Сучасні програмні рішення у області коворкінгів, часто орієнтовані на базові потреби користувачів - пошук коворкінгів за місцезнаходженням, доступністю простору та інфраструктурою. Однак, вони не здатні враховувати більш складні аспекти, такі як потреба в нетворкінгу або розвитку конкретних професійних навичок. Це, на мій погляд, є великим недоліком, оскільки це не враховує повний потенціал коворкінгів як місць для спільноти та професійного розвитку.

Щодо можливих шляхів покращення ситуації, інтеграція алгоритму підбору коворкінгів, який враховує нетворкінг та розвиток навичок, може в значній мірі покращити користувацький досвід. Користувачі зможуть отримувати рекомендації, які відповідають їх професійним потребам та цілям, замість простого пошуку за фізичними параметрами.

Для власників коворкінгів додаток також має велику цінність. Він не лише допомагає залучити нових відвідувачів, а й створює платформу для ефективного взаємодії з наявними клієнтами. Інформація про заходи та ресурси, доступні в коворкінгу, може бути легко доступна через додаток, дозволяючи власникам коворкінгів підвищити відвідуваність своїх подій та краще використати свої ресурси. Більше того, за допомогою алгоритму підбору, власники можуть точніше розуміти, які теми або ресурси є найбільш популярними серед користувачів, що може допомогти їм краще адаптувати свої пропозиції та плани на майбутнє.

У цьому контексті, було обрано шлях розробки кросплатформеного застосунку, який використовує інтелектуальний алгоритм для підбору коворкінгів на основі профілю користувача, його місцезнаходження, навичок, а також інформації про події та постійних відвідувачів коворкінгів. . Вибір кросплатформеного рішення відображає стратегію забезпечення максимальної доступності та корисності додатка для всіх користувачів, з метою забезпечити користувачам не просто доступ до інформації про коворкінги та можливість броні робочого місця, а допомогти їм максимально ефективно використовувати ці місця для реалізації своїх професійних цілей.

1.3 Аналіз існуючих технологій та успішних ІТ-проєктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації програмного забезпечення для управління та моніторингу роботи коворкінгів. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

В рамках розробки кросплатформеного застосунку для пошуку та рекомендації коворкінгів, важливо звернути увагу на відомі алгоритми

рекомендаційних систем і технічні рішення для створення веб- та мобільних застосунків.

Алгоритм рекомендаційної систем: Для розробки алгоритму рекомендаційної системи коворкінгів розглядалися різні алгоритми. Два з них найбільш відомі: Collaborative Filtering і Content-based Filtering.

Алгоритм рекомендації коворкінгу базується на використанні персональних даних користувача (навички, інтереси) та даних про коворкінги (заходи/воркшопи, відвідувачі) для створення ранжированого списку рекомендацій.

Такий підхід обрано з наступних причин:

– персоналізація: алгоритм надає рекомендації, що базуються на унікальних навичках та інтересах користувача, що робить рекомендації більш відповідними.

– актуальність: алгоритм враховує динаміку коворкінгів, включаючи проводження заходів/воркшопів.

– об'єктивність: алгоритм використовує статистичні дані про відвідувачів коворкінгу, що забезпечує об'єктивність рекомендацій.

– гнучкість: алгоритм може адаптуватися до змін даних користувача або коворкінгу, пропонуючи актуальні рекомендації.

– лематизація: використання лематизації допомагає згладити можливі розбіжності в термінології і покращити знаходження збігів.

Використовується гібридна модель рекомендаційної системи, що поєднує елементи Collaborative Filtering і Content-based Filtering.

Collaborative Filtering використовує інформацію про відвідування коворкінгів іншими користувачами зі схожими навичками і інтересами для генерації рекомендацій. Це допомагає знайти коворкінги, які можуть бути цікавими для користувача на основі вибору схожих користувачів.

Content-based Filtering використовує інформацію про заходи/воркшопи в коворкінгах, які збігаються з навичками та інтересами користувача. Це забезпечує релевантність рекомендацій з точки зору вмісту.

Комбінування цих двох підходів допомагає створити більш точну і гнучку рекомендаційну систему, забезпечує високу точність рекомендацій і персоналізований досвід для кожного користувача.

Технічні рішення для створення застосунку: Вибір архітектури застосунку є критичним кроком у розробці програмного забезпечення. Для цього проекту ми розглядали такі варіанти: Native Mobile Application Architecture, Cross-Platform Mobile Application Architecture, та Hybrid Mobile Application Architecture.

Native Architecture має перевагу високої продуктивності та оптимізації під конкретну платформу, але вона вимагає розробки окремого коду для кожної платформи (iOS/Android), що збільшує час та ресурси на розробку.

Cross-Platform Architecture дозволяє використовувати один кодовий базис для обох платформ, але має певні обмеження у використанні нативних функцій і може мати проблеми з продуктивністю.

Hybrid Mobile Application Architecture, яку ми обрали для нашого проекту, поєднує переваги обох вищезазначених архітектур. Вона дозволяє використовувати один кодовий базис для обох платформ, включаючи використання веб-технологій, та має доступ до нативних функцій через використання нативних оболонок. Це значною мірою спрощує процес розробки та підтримки, забезпечуючи при цьому хорошу продуктивність та користувацький досвід. Для реалізації цієї архітектури ми обрали фреймворк Ionic [6], який надає широкий набір інструментів та можливостей для гібридної розробки.

1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

У процесі розробки проекту було використано ряд допоміжних програмних засобів, фреймворків та платформ. Підбір цих інструментів здійснювався на основі їх здатності задовольнити потреби проекту.

Для клієнтської частини проекту основним засобом розробки було обрано VSCode [9], який є універсальним редактором коду з можливістю розширення за допомогою плагінів. Він підтримує багато мов програмування, включаючи TypeScript, який використовувалися у даному проекті. Основними фреймворками для клієнтської частини стали Ionic та React [8]. Ionic - це фреймворк для розробки кросплатформених додатків, який надає широкі можливості створення нативного досвіду користувача на різних платформах. React був використаний для створення користувацького інтерфейсу за допомогою компонентного підходу. Інструмент Capacitor [7] був використаний для компіляції веб-додатку під Android/iOS пристрої та інтеграцією з нативними функціями пристроїв. Для компілювання та тестування Android-версії додатку використовувалась Android Studio [10].

Серверна частина проекту була розроблена за допомогою Firebase [5] Functions на основі Node.js — середовище виконання JavaScript, яке дозволяє створювати на цій мові програмування серверні API. Firebase Authentication модуль був вибраний для авторизації користувачів у системі через його широкі можливості і простоту впровадження.

Для зберігання та обробки даних використовувалася база даних Firebase DB**.** Firebase DB відповідає за зберігання та обробку даних, а також надає інтерфейс для взаємодії з цими даними.

Таким чином, вибір допоміжних програмних засобів та засобів розробки був здійснений на основі їх функціональності, та потреб проекту.

1.3.3 Аналіз відомих програмних продуктів

Для порівняння застосунків пошуку та бронювання коворкінгів було обрано два з найпопулярніших програмних продуктів в цій області - LiquidSpace та WeWork.

LiquidSpace - застосунок ринку офісних просторів на вимогу. Можна бронювати робочі місця на години, дні або на довший період. Крім того, LiquidSpace надає інструменти для управління гібридними робочими місцями. Інтерфейс користувача в застосунку має наступний вигляд:

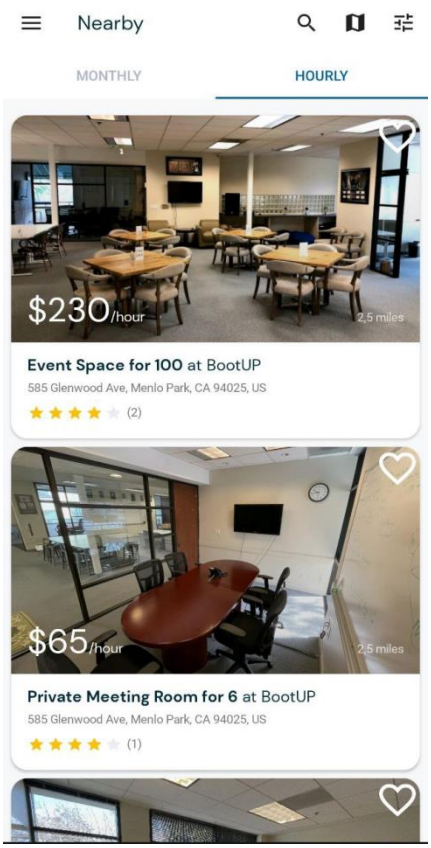


Рисунок 1.11 – Інтерфейс користувача в застосунку LiquidSpace

Переваги:

– надає широкий вибір фільтрів пошуку просторів для роботи;
– дозволяє здійснити бронювання в реальному часі.

Недоліки:

– відсутність персоналізованого рекомендаційного пошуку;
– відсутність можливостей нетворкінгу;
– відсутність підбору робочого місця в коворкінгах.

WeWork - це застосунок з оренди офісних просторів, яка надає спільну робочу зону для своїх клієнтів. Щоб скористатися послугами WeWork, вам потрібно зареєструватися на їх веб-сайті.

Інтерфейс користувача в застосунку має наступний вигляд:

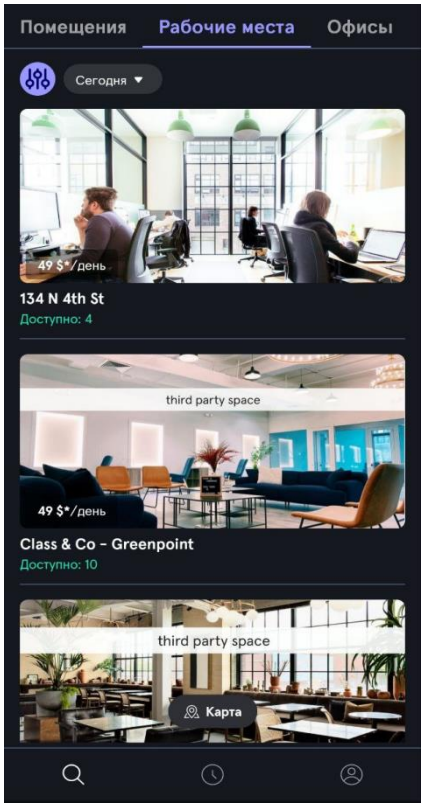


Рисунок 1.12 – Інтерфейс користувача в застосунку WeWork

Переваги:

– надає доступ до глобальної мережі коворкінгів;
– можливість бронювання зручним способом.

Недоліки:

– сервіс в основному спрямований на довготривале бронювання;
– відсутність персоналізованих рекомендацій;
– відсутність можливостей нетворкінгу;
– відсутність підбору робочого місця в коворкінгах.

Програмний продукт має перевагу перед зазначеними аналогами через наступні особливості:

– персоналізовані рекомендації — використовуючи алгоритми рекомендаційних систем, застосунок забезпечує персоналізований досвід для кожного користувача;
– можливості нетворкінгу — застосунок дозволяє користувачам розширювати свої професійні зв’язки, завдячуючи рекомендаційному алгоритму, який пропонує найперспективніші варіанти;
– гнучкість бронювання — незалежно від того, шукаєте ви довготривале місце для роботи чи кілька годин, застосунок надає гнучкі опції бронювання, що відповідають вашим потребам;

Таким чином, застосунок розроблено з метою удосконалення слабких сторін сучасних аналогів та надання кращого досвіду користувачам.

1.4 Аналіз вимог до програмного забезпечення

Головною функцією програмного забезпечення є надання високоякісного, ефективного та персоналізованого досвіду пошуку та бронювання коворкінгів, більше функцій можна побачити на рисунку 1.2.

Рисунок 1.2 – Діаграма варіантів використання

В таблицях 1.11 - 1.21 наведені варіанти використання програмного забезпечення.

Таблиця 1.11 - Варіант використання UC-1

Use case name	Реєстрація користувача /адміністратора коворкінгу
Use case ID	UC-01
Goals	Реєстрація нового користувача /адміністратора коворкінгу в системі
Actors	Гість (незареєстрований користувач)
Trigger	Користувач бажає зареєструватися
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку реєстрації. В поля для реєстрації вводяться відповідні дані: пошта користувача, пароль в системі, та його повтор для підтвердження, а також чек бокс для підтвердження умов сервісу. Після заповнення даних користувача натискає кнопку реєстрації. Після цього з'являється повідомлення про успішну реєстрацію, і користувач перенаправляється на сторінку входу. Якщо обирається реєстрація нового користувача то додатково є поля навичок для заповнення.
Extension	У випадку введення не коректних даних, кнопка реєстрації стає неактивною. Якщо якесь конкретне поле введено некоректно, то воно підсвічується червоним надписом.
Post-Condition	Створення сторінки користувача, перехід на сторінку входу. Для адміністратора реєстрація продовжується на сторінці збору даних про коворкінг

Таблиця 1.12 - Варіант використання UC-2

Use case name	Авторизація користувача/ адміністратора коворкінгу
Use case ID	UC-02
Goals	Автентифікація користувача в системі
Actors	Зареєстрований користувач
Trigger	Користувач бажає ввійти до системи
Pre-conditions	Користувач зареєстрований
Flow of Events	Користувач запускає застосунок, вводить пошту та пароль за допомогою яких він реєструвався і натискає кнопку входу.
Extension	У випадку спроби входу користувачем за допомогою невірних даних, користувач отримає повідомлення про неможливість входу, та причину, наприклад по тій причині, що він використав електронну пошту якої немає в базі даних користувачі
Post-Condition	Для користувача перехід на екран пошуку коворкінгів / для адміністраторів на сторінку бронювання робочих місць

Таблиця 1.13 - Варіант використання UC-3

Use case name	Збір даних про коворкінг
Use case ID	UC-03
Goals	Збір даних про коворкінг для реєстрації коворкінгу
Trigger	Користувач ввійшов до застосунку вперше
Pre-conditions	Після реєстрації та успішної автентифікації, користувачу необхідно внести дані про коворкінг
Flow of Events	Після того як представник коворкінгу ввійшов до застосунку вперше, йому необхідно ввести дані, а саме: назву коворкінгу, локацію, опис та додати фото.
Extension	
Post-Condition	Перехід на екран бронювання робочих місць

Таблиця 1.14 - Варіант використання UC-4

Use case name	Адміністратор коворкінгу бажає вказати доступний час робочим місцям в коворкінгу
Use case ID	UC-04
Goals	Задання робочим місцям вільного часу для їх можливого бронювання
Actors	Адміністратор коворкінгу
Trigger	Адміністратор коворкінгу бажає задати вільний час для робочих місць
Pre-conditions	Адміністратор коворкінгу зареєстрував коворкінг та додав робочі місця
Flow of Events	Адміністратор коворкінгу переходить на сторінку бронювання. Виділяє одне робоче місце або декілька та натискає кнопку вказати вільний час. Вказує дату та час в відповідні поля. Після заповнення даних адміністратор/ представник натискає кнопку підтвердити. Після цього по цим робочим місцям з’являється відображення вільного часу.
Extension	У випадку не підтвердження, введені дані очищуються.
Post-Condition	Відображення коворкінгу в пошуку для користувачів.

Таблиця 1.15 - Варіант використання UC-5

Use case name	Адміністратор коворкінгу бажає додати робоче місце, яке зможуть забронювати користувачі на певну дату і час
Use case ID	UC-05
Goals	Додавання робочого місця для свого коворкінгу
Actors	Адміністратор коворкінгу
Trigger	Адміністратор бажає додати робоче місце для можливості бронювання його користувачами застосунку
Pre-conditions	Адміністратор коворкінгу зареєструвався та додав інформацію про коворкінг
Flow of Events	Адміністратор переходить на сторінку робочих місць. Натискає на кнопку додавання робочого місця. В поля для додавання робочого місця вводяться відповідні дані: відносну позицію в коворкінгу, ціну бронювання за годину, кількість місць. Після заповнення даних адміністратор натискає на кнопку підтвердження на цій картці та місце додається в базу даних.
Extension	
Post-Condition	Відображення доданої робочого місця на цій сторінці для адміністратора та на сторінці коворкінгу бронювання місць для користувача.

Таблиця 1.16 - Варіант використання UC-6

Use case name	Користувач бажає знайти та обрати коворкінг змінюючи параметри
Use case ID	UC-06
Goals	Знайти та обрати найкращий варіант коворкінгу
Actors	Користувач (зареєстрований)
Trigger	Користувач зареєструвався та перейшов на сторінку пошуку коворкінгів
Pre-conditions	Користувач зареєструвався
Flow of Events	Користувач зареєструвався (ввів свою локацію на екрані реєстрації) або авторизувався. Він переходить на сторінку пошуку коворкінгів, де йому одразу пропонуються певні варіанти. Користувач хоче змінити параметри та більше персоналізувати пошук — натискає на кнопку пошуку, відкривається діалогове вікно де користувач змінює і локацію і свої навички або щось одне. Після виходу з діалогового вікна результати рекомендацій коворкінгів можуть змінитися.
Extension	В випадку введення неправильної локації, минуле значення локації не змінюється.
Post-Condition	Вихід з діалогового вікна, відображення нових результатів пошуку.

Таблиця 1.17 - Варіант використання UC-7

Use case name	Користувач хоче забронювати місце в обраному коворкінгу
Use case ID	UC-07
Goals	Бронювання місця в обраному коворкінгу
Actors	Користувач (зареєстрований)
Trigger	Користувач натискає на обраний коворкінг на сторінці пошуку
Pre-conditions	Користувач зареєструвався/ авторизувався
Flow of Events	На сторінці пошуку коворкінгів користувач обрав свій варіант та натиснув на нього. Користувач переходить на сторінку бронювання місця в коворкінгу. На цій сторінці користувач бачить назву, опис, локацію та картинки коворкінгу. Вказує дату, та бачить доступні робочі місця. Обирає місце по характеристиці обирає доступну дату, час, та натискає кнопку забронювати.
Extension	
Post-Condition	Створюється бронювання робочого місця, яке відображається на сторінці бронювань.

Таблиця 1.18 - Варіант використання UC-8

Use case name	Адміністратор хоче додати бронювання робочого місця у своєму коворкінгу
Use case ID	UC-08
Goals	Бронювання місця в обраному своєму коворкінгу, від можливих сторонніх клієнтів
Actors	Користувач (зареєстрований)
Trigger	Адміністратор виділяє місце (місця), та натискає на кнопку бронювання
Pre-conditions	Адміністратор зареєструвався/ авторизувався, додав опис коворкінгу а також робоче місце
Flow of Events	На сторінці робочих місць свого коворкінгу адміністратор виділив робоче місце (місця) та натиснув кнопку бронювання. Адміністратору відкривається модальне вікно на якому він має обрати дату або декілька дат, задати часовий період обраної дати, обрати опцію Booked із двох можливих опцій. Далі адміністратор натискає кнопку Confirm — запис бронювання від адміністратора створюється в БД
Extension	Якщо адміністратор закриває модальне вікно або натискає на кнопку Cancel — бронювання не створюється.
Post-Condition	Створюється бронювання робочого місця, яке відображається на сторінці бронювань.

Таблиця 1.19 - Варіант використання UC-9

Use case name	Користувач хоче побачити інформацію про свої бронювання
Use case ID	UC-09
Goals	Користувач бачить свої актуальні бронювання, історію своїх бронювань та переглядає детальну інформацію про коворкінг у якому забронював місце
Actors	Користувач (зареєстрований)
Trigger	Користувач натискає на кнопку меню і обирає опцію Bookings для переходу на сторінку усіх своїх бронювань.
Pre-conditions	Користувач зареєструвався/ авторизувався
Flow of Events	Користувач забронював робоче місце в коворкінгу або не бронював. На сторінці пошуку коворкінгу або на сторінці бронювання робочих місць в коворкінгу користувач відкриває меню та обирає сторінку Bookings. На цій сторінці користувач бачить свої актуальні бронювання та усі минулі бронювання, які завершились успішно або були скасовані. По кожному своєму бронювання користувач бачить дату, час і статус, а також кнопки-іконки коворкінгів. Користувач натискає на кнопку-іконку та здійснює перехід на сторінку бронювання робочого місця обраного коворкінгу.
Extension	
Post-Condition	Перехід користувача на сторінку бронювання робочого місця по обраному коворкінгу у якому до цього бронював відповідне місце

Таблиця 1.20 - Варіант використання UC-10

Use case name	Користувач хоче скасувати актуальне бронювання
Use case ID	UC-10
Goals	Користувач бачить свої актуальні бронювання та історію своїх бронювань (та відміняє бронювання)
Actors	Користувач (зареєстрований)
Trigger	Користувач натискає на кнопку меню і обирає опцію Bookings для переходу на сторінку усіх своїх бронювань. На цій сторінці біля відповідного бронювання натискає на кнопку-іконку скасування
Pre-conditions	Користувач зареєструвався/ авторизувався та забронював робоче місце на відповідну дату та час
Flow of Events	Користувач забронював робоче місце в коворкінгу. Далі він переходить на сторінку своїх бронювань. На цій сторінці користувач бачить свої актуальні бронювання та усі минулі бронювання, які завершились успішно або були скасовані. Якщо користувач хоче скасувати актуальне бронювання — він поряд з обраним бронюванням натискає на кнопку-іконку хрестик. У діалоговому вікні користувач підтверджує відміну.
Extension	Якщо користувач у діалоговому вікні не підтверджує відміну бронювання, то нічого не скасовується.
Post-Condition	Бронювання скасовується , а часовий слот бронювання стає доступний іншим користувачам.

Таблиця 1.21 - Варіант використання UC-11

Use case name	Адміністратор коворкінгу хоче скасувати актуальне бронювання на обраному робочому місці у своєму коворкінгу
Use case ID	UC-11
Goals	Адміністратор скасовує бронювання робочого місця, після чого відповідний часовий слот стає доступним для повторного бронювання.
Actors	Адміністратор коворкінгу
Trigger	Адміністратор біля відповідного бронювання картки робочого місця натискає на кнопку скасування
Pre-conditions	Адміністратор авторизувався
Flow of Events	Користувач забронював робоче місце в коворкінгу. Далі він переходить на сторінку своїх бронювань. На цій сторінці користувач бачить свої актуальні бронювання та усі минулі бронювання, які завершились успішно або були скасовані. Адміністратор натискає на кнопку скасування актуального бронювання. У діалоговому вікні користувач підтверджує відміну.
Extension	Якщо користувач у діалоговому вікні не підтверджує відміну бронювання, то нічого не скасовується.
Post-Condition	Бронювання скасовується , а часовий слот бронювання стає доступний іншим користувачам.

1.4.1 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. На рисунку 1.4 наведено загальну модель вимог, а в таблиці 1.2 описано функціональні вимоги до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 1.5.

№	Назва функціональної вимоги	ID вимоги
1	Система авторизації	FR-1
1.1	Авторизація користувача та адміністратора	FR-2
1.2	Реєстрація користувачів та адміністраторів	FR-3
2	Збереження та оновлення даних коворкінгів	FR-4
3	Автоматичне оновлення статусів бронювань у БД	FR-5
4	Обробка бронювань місць у коворкінгах	FR-6
5	Обробка даних та персоналізований алгоритм рекомендацій	FR-7
5.1	Зміна пошукових даних алгоритму	FR-8
6	Задання доступних дат та часових слотів для робочих місць	FR-9
7	Зміна записів та перегляд історії	FR-10

Рисунок 1.4 – Модель вимог у загальному вигляді

Таблиця 1.2– Функціональні вимоги FR

Номер	Назва	Опис
FR-1	Система авторизації	Система повинна надавати можливість реєстрації та авторизації користувачів та адміністраторів коворкінгів шляхом введення пошти, паролю
FR-2	Авторизація користувача та адміністратора	Користувач та адміністратор повинен мати можливість авторизуватись у системі
FR-3	Реєстрація користувача та адміністратора коворкінгу	Користувач та адміністратор повинен мати можливість зареєструватись у застосунку
FR-4	Збереження та оновлення даних коворкінгів	Застосунок повинен надавати можливість адміністраторам зберігати та оновлювати дані коворкінгів, робочі місця в них, доступність та характеристику цих місць у БД
FR-5	Автоматичне оновлення статусів бронювань у БД	Кожну годину база даних повинна оновлювати статуси бронювань відповідно до теперішнього часу
FR-6	Обробка бронювань місць у коворкінгах	Система повинна відповідно реагувати на нові бронювання та їх скасування
FR-7	Обробка даних та персоналізований алгоритм рекомендацій	Застосунок повинен пропонувати найперспективніші з точки зору нетворкінга варіанти коворкінгів для користувачів
FR-8	Зміна пошукових даних алгоритму	Користувач має змогу змінювати дані, які використовуються для рекомендаційного алгоритму

Продовження таблиці 1.2

FR-9	Задання доступних дат та часових слотів для робочих місць	Застосунок має надавати можливість адміністраторам задавати доступні дати та часові слоти для бронювання робочих місць, а також відповідно відображати цю доступність користувачам
FR-10	Зміна записів та перегляд історії	Застосунок має надавати можливість відміни бронювання користувачам та надавати повну історію їх бронювань

Варіанти використання є похідними від функціональних вимог, по результату трасування маємо наступну таблицю на рисунку 1.5

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10
UC-1	+									
UC-2		+								
UC-3			+							
UC-4			+	+		+				
UC-5				+	+		+		+	
UC-6			+				+	+		
UC-7				+		+			+	+
UC-8						+			+	+
UC-9			+		+			+	+	
UC-10				+	+		+			+
UC-11		+				+		+	+	

Рисунок 1.5 – Таблиця трасування вимог

1.4.2 Розроблення нефункціональних вимог

Нефункціональні вимоги характеризують якість системи, її поведінку та обмеження, які вона повинна виконувати. До застосунку було висунуто наступні вимоги:

– швидкість — час відклику системи не повинен перевищувати 5 секунд для основних дій користувача, таких як вхід у систему, перегляд та бронювання коворкінгів;

– надійність — система повинна працювати стабільно без помилок протягом тестового періоду. Процес відновлення повинен бути простим у випадку системних збоїв;

– безпека — персональні дані користувачів повинні зберігатися в безпечному форматі. Система повинна використовувати базові методи аутентифікації для захисту від несанкціонованого доступу;

– зручність використання — інтерфейс системи повинен бути простим і зрозумілим, що дозволить користувачам легко виконувати основні функції;

1.5 Постановка задачі

Метою даного проекту є підвищення ефективності організації робочих процесів та зниження трудомісткості управління ресурсами коворкінгів. Метою розробки є створення універсального інструменту, що полегшить управління коворкінгами, забезпечить прозорість використання простору, автоматизує адміністративні процеси та сприятиме ефективності використання ресурсів. Розробка спрямована на забезпечення зручного досвіду користування для керівників коворкінгів, їх персоналу, а також відвідувачів та користувачів цих просторів.

Задачами, які підлягають розв'язанню в ході роботи над проектом, є:

– проаналізувати сучасні програмні рішення в цій області та визначити їх сильні та слабкі сторони;

– розробити архітектуру та дизайн системи, в тому числі бази даних, серверну та клієнтську частини;

– реалізація функціоналу управління ресурсами коворкінгів, пошуку та бронювання робочих місць;

– розробка та інтеграція рекомендаційного алгоритму з перспективою нетворкінгу;

– розробка стратегії тестування та виконання тестових випробувань.

Висновки до розділу

В даному розділі було проведено аналіз предметної області, поставлено задачі розробки кросплатформеного застосунку. Розглянуто та обумовлено основні функціональні та нефункціональні вимоги до додатку, такі як висока продуктивність, надійність, простота використання, надання персоналізованих рекомендацій тощо.

Було визначено, що кросплатформена архітектура за допомогою фреймворку Ionic з використанням Capacitor для інтеграції з нативними функціями пристроїв буде найкращим рішенням для даного проекту, оскільки вона дозволить оптимізувати процес розробки та підтримки додатку.

В рамках цього розділу також було обрано середовище розробки - VSCode, яке підтримує використовувані мови програмування та має великий набір плагінів для підвищення продуктивності розробки.

Що стосується серверної частини, то було прийнято рішення про використання Node.js у сполученні з сервісами Firebase - Firebase Functions для реалізації серверної логіки, Firebase Authentication для авторизації користувачів та Firebase DB для зберігання та обробки даних.

Результатом аналізу є чітке формулювання завдань розробки, вибір технологій для реалізації проекту та побудова базової концепції архітектури додатку. Даний аналіз став основою для переходу до наступного етапу - проектування системи.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Для опису бізнес процесу програмного забезпечення використовується BPMN модель (рисунок 2.1).

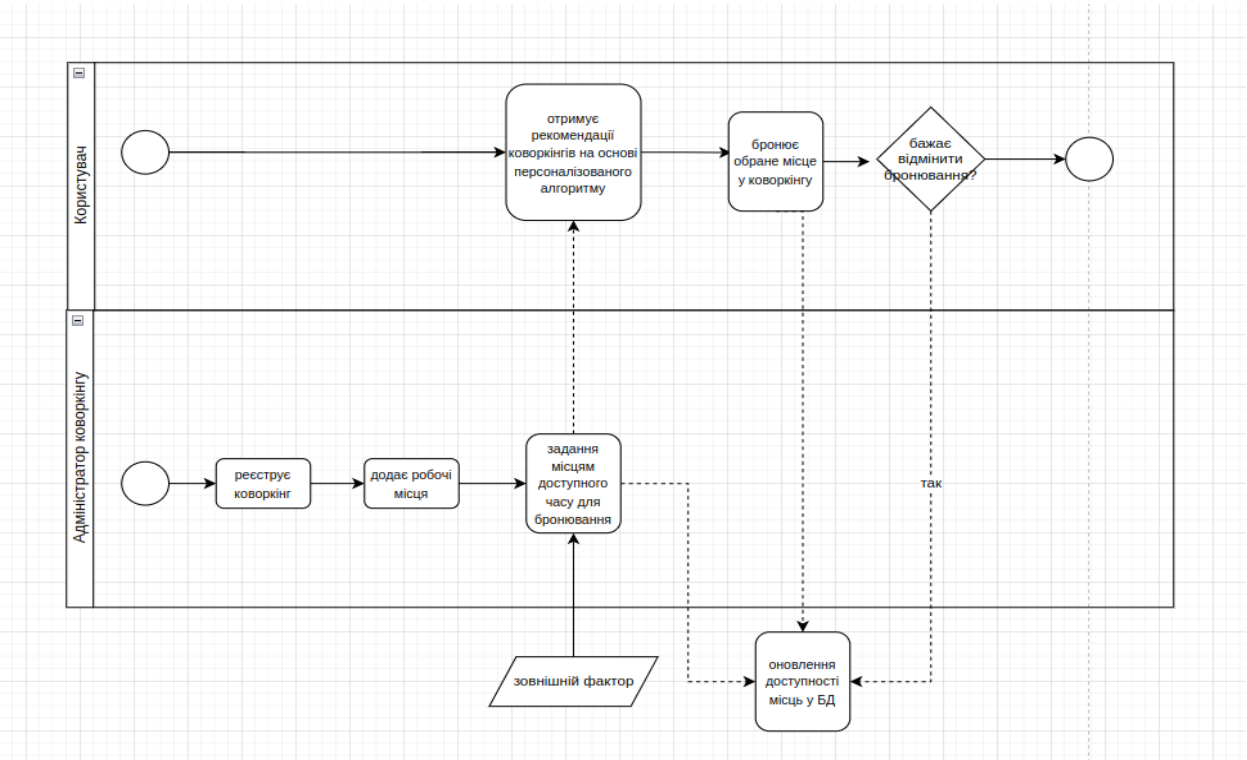


Рисунок 2.1 – Схема бізнес-процесу

Опис послідовності бронювання робочого місця користувачем:

– зареєстрований / автентифікований користувач на сторінці пошуку коворкінгів отримує рекомендації коворкінгів на основі персоналізованого алгоритму;

– користувач обирає коворкінг та бронює робоче місце на заплановані дату та час;

– у разі якщо плани користувача змінились він має можливість скасувати бронювання;

Опис послідовності зміни доступності робочого місця адміністратором коворкінгу:

– адміністратор коворкінгу вносить дані спочатку про сам коворкінг;

– далі додає робочі місця, яким після чого зможе задавати доступний час;

– адміністратор задає робочим місцям доступний час для бронювання;

– також є зовнішній фактор, адміністратору потрібна можливість додавати бронювання робочим місцям свого коворкінгу.

2.2 Архітектура програмного забезпечення

У даному проєкті прийнято рішення застосувати архітектуру, яка включає в себе BaaS (Backend as a Service) та кросплатформений підхід розробки. Це забезпечує гнучкість, продуктивність та швидкість розробки, а також можливість використовувати один і той же код для різних платформ.

BaaS (Backend as a Service) архітектура. Використання BaaS як Firebase дозволяє перемістити навантаження пов'язане з розробкою та управлінням серверної частини на сторону постачальника. До цього відносяться робота з базою даних, рекомендаційний алгоритм, авторизація користувачів, а також реалізація різноманітних серверних функцій, які будуть розміщені в Firebase Functions.

Переваги використання BaaS архітектури:

– зменшення часу на розробку за рахунок використання готових служб на серверній частині, які можуть бути швидко інтегровані в додаток, готові до використання служби серверної частини;

– масштабованість: BaaS платформи автоматично масштабуються для обробки збільшення або зменшення навантаження, що забезпечує надійність та ефективність;

– зменшення часу розробки; BaaS забезпечує готові до використання служби на серверній частині, які можуть бути швидко інтегровані в додаток, що призводить до значного скорочення часу розробки;

– безпека; BaaS провайдери зазвичай включають у свої рішення безпеку даних, автентифікацію користувачів і забезпечення конфіденційності, як наприклад у нашому випадку Firebase.

Кросплатформена архітектура з web-first підходом. В основі підходу "web-first" лежить ідея створення веб-застосунку, який потім адаптується для мобільних платформ. Завдяки використанню Ionic, ми можемо створювати один код, який має можливості розгортуватися на різних платформах (веб, Android, iOS). Разом з Ionic архітектуру кросплаформи забезпечує інструмент Capacitor — це середовище виконання, який дозволяє запускати код на різних платформах, що використовується в Ionic для створення нативних бінарних файлів з веб-коду.

Web-first підхід має декілька ключових переваг:

– консистентність інтерфейсу — використовуючи webview, інтерфейс Ionic дозволяє зберігати консистентність відображення на різних платформах;

– швидкість розробки — код веб-застосунку можна використати для створення мобільного додатка, що зменшує кількість роботи, час та витрати на розробку;

– легкість оновлення — оновлення веб-застосунка відбувається миттєво, без необхідності чекати на затвердження в магазинах додатків;

– більша доступність — веб-застосунки можна використовувати на будь-якому пристрої з браузером, що забезпечує більший охоплення користувачів.

Архітектура застосунку в загальному вигляді представлена на рисунку 2.2

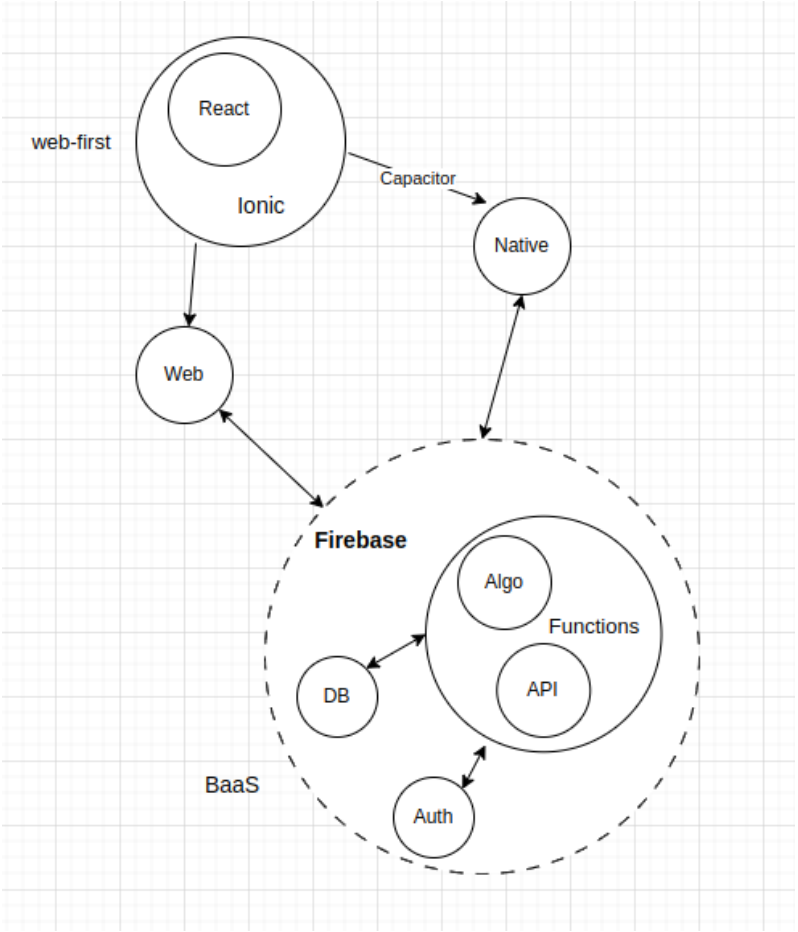


Рисунок 2.2 – Архітектура застосунку в загальному вигляді

По модульно архітектуру програмного забезпечення можна розділити на наступні частини: модулі клієнтської частини (Registration, <=> Login modules; Search coworkings module => Book working places module => Bookings module; Add coworking information module => Manage working places module => Working places availability module); модулі серверної частини як частина Firebase Function (Structure Coworkings module => Networking algorithm => Location third party; Bookings status auto-update). Нижче на рисунку 2.3 представлена діаграма архітектури ПЗ.

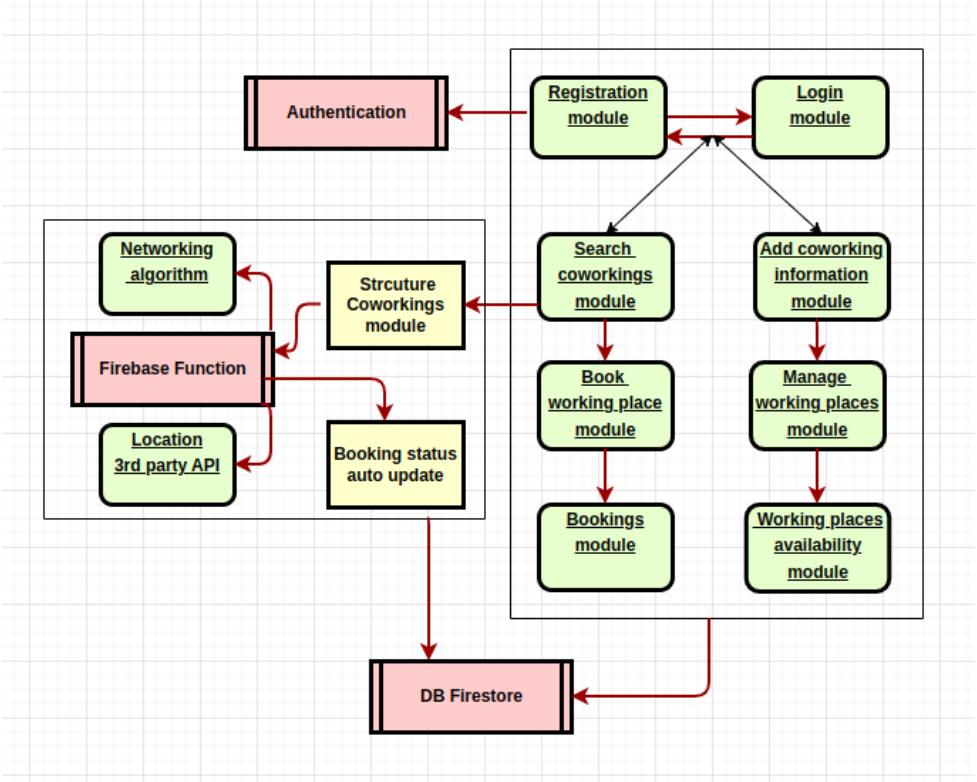


Рис. 2.3 — Архітектура програмного забезпечення

Далі наведено короткий опис по кожному модулю (частині) застосунку, його призначення і методи.

Registration and Login Modules

– Призначення: Управління процесом реєстрації та авторизації користувачів.

– Методи: Реєстрація (створення нового облікового запису), вхід (перевірка ідентифікатора користувача і пароля), відновлення пароля.

Search Coworkings and Book Working Places Modules

– Призначення: Забезпечує можливість пошуку коворкінгів та бронювання робочих місць.

– Методи: Пошук (фільтрує доступні коворкінги на основі вказаних критеріїв), бронювання (створює новий запис бронювання).

Bookings Module

– Призначення: Управління інформацією про бронювання користувача.

– Методи: Перегляд бронювань (показує всі поточні і минулі бронювання користувача), відміна бронювань.

Add Coworking Information and Manage Working Places Modules

– Призначення: Дозволяє коворкінгам додавати та редагувати інформацію про себе, управляти доступністю робочих місць.

– Методи: Додати/оновити інформацію (створює/оновлює обліковий запис коворкінгу), управління робочими місцями (створення, видалення, редагування робочих місць).

Working Places Availability Module

– Призначення: Надає інформацію про доступність робочих місць в режимі реального часу.

– Методи: Показ доступності (відображення статусу робочого місця - вільно/зайнято).

Structure Coworkings Module, Networking Algorithm and Location Third Party (Firebase Function)

– Призначення: Забезпечує функціонал сервісу з сторони сервера, включаючи структурування даних коворкінгів, алгоритми пошуку та взаємодію зі сторонніми сервісами геолокації.

– Методи: Обробка даних (формує структуру даних коворкінгів), пошук (виконує алгоритми, що допомагають знайти коворкінги на основі критеріїв), геолокація (взаємодія зі сторонніми сервісами для отримання даних про геолокацію).

Bookings Status Auto-update Module (Firebase Function)

– Призначення: Автоматично оновлює статус бронювань, заснований на часі.

– Методи: Автооновлення (перевірка всіх активних бронювань і оновлення їх статусу відповідно до поточного часу).

2.3 Конструювання програмного забезпечення

У цьому пункті детально розглянуто які технології та інструменти було застосовано при конструюванні застосунку.

2.3.1 Вибір інструментів для написання клієнтської частини застосунку

Для написання клієнтської частини було обрано Ionic framework на mobile-friendly компонентах, а також Capacitor для утворення з веб-застосунку нативних мобільних версій, використовуючи бібліотеку React, як основу і Typescript як надійну надбудову JS.

Ionic - це один з найбільш популярних фреймворків для розробки крос-платформних мобільних додатків. Він використовує веб-технології, для створення додатків, що виглядають і працюють як нативні. Ionic має багатий набір готових компонентів, які забезпечують приємний користувацький інтерфейс та забезпечують високий рівень продуктивності. Також, використовуючи Ionic, ви можете написати код один раз і запустити його на будь-якій платформі, що дозволяє значною мірою економити час та ресурси.

Capacitor є сучасним способом створення нативних мобільних додатків з використанням веб-технологій. Цей інструмент був створений командою Ionic і дозволяє вам використовувати одну і ту ж базу коду для веб, iOS і Android. Capacitor надає доступ до нативних API на рівні JavaScript, що дозволяє використовувати всю потужність нативних платформ з комфортом веб-розробки.

React - це бібліотека JavaScript, яка була розроблена Facebook для побудови інтерфейсів користувача. React надає високий рівень гнучкості та ефективності завдяки своєму віртуальному DOM і системі компонентів. Це покращує продуктивність додатку і спрощує розробку та супровід.

TypeScript - це супер-сет JavaScript, який додає статичну типізацію, що значно покращує продуктивність розробки та якість коду. Використання TypeScript допомагає запобігти багато помилкам на етапі компіляції, що дозволяє вам писати більш надійний код. TypeScript також має великий набір інструментів, що дозволяє вам писати більш чистий і організований код.

Загалом, ці технології були обрані за їх ефективність, гнучкість, підтримку спільноти і широкі можливості для розробки якісних мобільних додатків.

2.3.2 Вибір інструментів для написання серверної частини застосунку

Було прийнято рішення використовувати інфраструктуру Firebase для написання серверної частини застосунку, а саме: Authentication, Firestore, Hosting та Function.

Firebase Authentication був обраний для аутентифікації користувачів, так як він пропонує просте, безпечне та ефективне рішення для управління користувачами. Він надає широкий спектр опцій аутентифікації, включаючи електронну пошту і пароль, телефонну верифікацію, Google, Facebook та інші.

Firebase Firestore використовується для управління базою даних в реальному часі. Це гнучке, масштабоване та повністю кероване рішення для зберігання та синхронізації даних, яке забезпечує високий рівень продуктивності та надійності.

Firebase Hosting було обрано для розгортання та хостингу застосунку в Інтернеті. Він надає швидке, надійне та безпечне розгортання статичного та динамічного контенту.

Firebase Function, що використовує 2 gen firebase-function Node modular approach, було обрано для створення серверних функцій. Це дозволяє писати код, який буде виконуватись у відгук на певні події в Firebase та інших сервісах в обліковому записі.

Для отримання координат по локаціям обрано geocoding API, яке надає надійний спосіб конвертації адрес в географічні координати.

Нарешті, для автоматичного оновлення статусу бронювань було обрано node-cron. Це дозволяє створювати задачі, які виконуються за певним графіком, що значно спрощує процес моніторингу та оновлення даних.

2.3.3 База даних

Firebase пропонує гнучкий та користувацький інтерфейс, який дозволяє легко зберігати та управляти даними, він ідеально підходить для роботи з JS React.

Ця платформа забезпечує миттєве оновлення даних у режимі реального часу завдяки її NoSQL структурі, що становить велику вагу для додатків, які вимагають негайних оновлень.

Firebase також надає масштабованість, що дозволяє без проблем адаптуватися до зростання додатку, та інтегровані рішення для автентифікації, резервного копіювання, аналітики та повідомлень.

Тому, використовуючи Firebase, можна зосередитися на оптимізації функціональності додатку, замість управління серверною інфраструктурою.

Зважаючи на ці ключові аспекти Firebase був обраний в якості бази даних для застосунку. Схему бази даних зображено на рисунку 2.4:

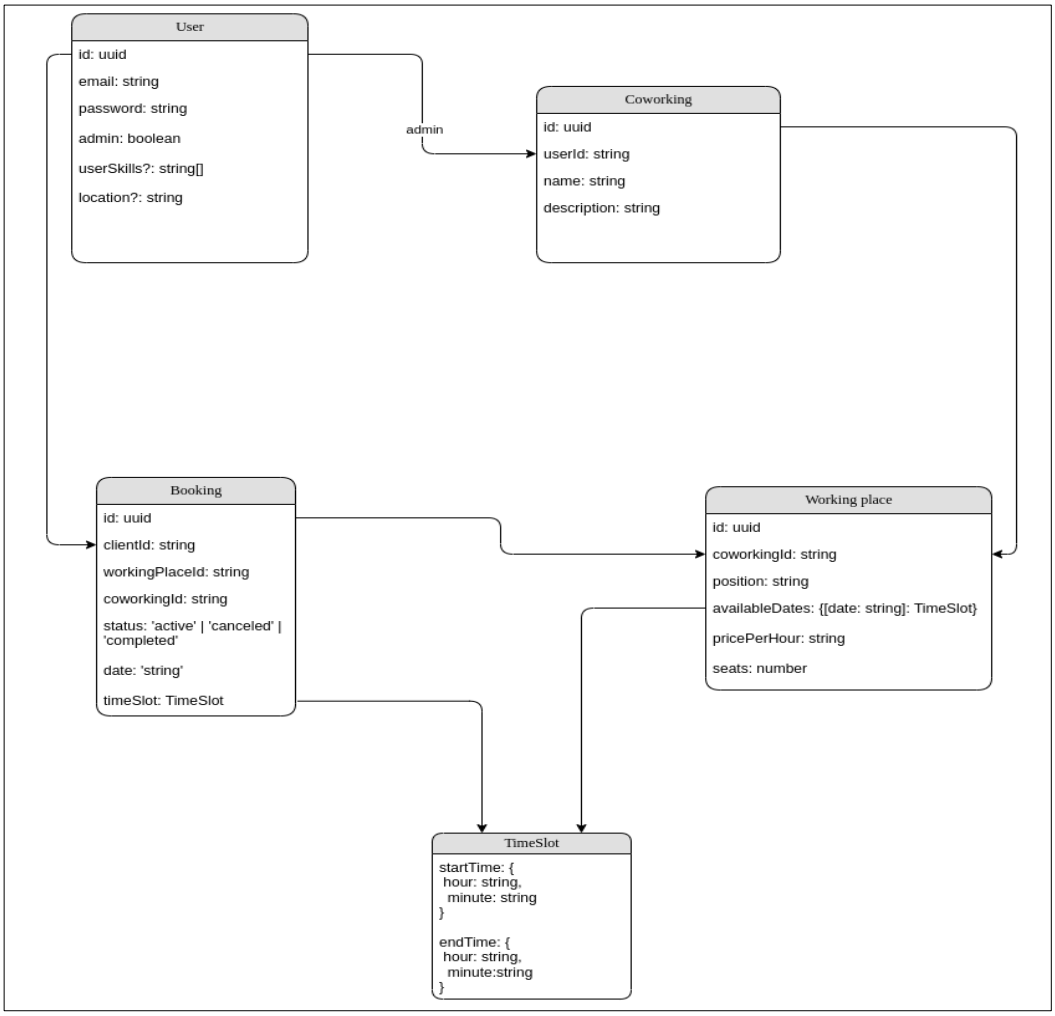


Рисунок 2.4 – Схема бази даних

В якості системи управління базами даних використовується Firebase Firestore. Опис сутностей бази даних наведено у таблицях 2.11 - 2.15.

Таблиця 2.11 – Опис таблиці Client

Таблиця	Назва поля	Тип даних	Опис
Client	id	uuid	Ідентифікаційний номер користувача
	email	string	Електронна пошта користувача
	password	string	Пароль користувача
	userSkills	string[] or undefined	Якщо не адміністратор коворкінгу то має бути масив навичок
	admin	boolean	Чи є користувач адміністратором коворкінгу?
	location	string or undefined	Якщо не адміністратор то задається остання введена локація

Таблиця 2.12 – Опис сутності TimeSlot

сутність	Назва поля	Тип даних	Опис
TimeSlot	startTime	{ hour: string, minute: string }	Початок часово проміжку (слоту)
	endTime	{ hour: string, minute: string }	Кінець часового проміжку

Таблиця 2.13 – Опис таблиці Coworking

Таблиця	Назва поля	Тип даних	Опис
Coworking	id	uuid	Ідентифікаційний номер коворкінгу
	userId	string	Ідентифікаційний номер адміністратора коворкінгу
	name	string	Назва коворкінгу
	description	string	Опис
	location	string	Локація (місто, район)

Таблиця 2.14 – Опис таблиці Working place

Таблиця	Назва поля	Тип даних	Опис
Working place	id	uuid	Ідентифікаційний номер робочого місця
	coworkingId	string	Ідентифікаційний номер коворкінгу
	position	string	Відносна позиція в коворкінгу
	availableDates	string	Доступні часові слоти для бронювання на вказані дати
	pricePerHour	string	Ціна за годину оренди
	seats	number	Кількість місць

'Таблиця 2.15 – Опис таблиці Booking

Таблиця	Назва поля	Тип даних	Опис
Booking	Id	uuid	Ідентифікатор бронювання
	clientId	string	Ідентифікаційний номер клієнта
	workplaceId	string	Ідентифікаційний номер робочого місця
	coworkingId	string	Ідентифікаційний номер коворкінгу
	timeSlot	string	Часовий слот
	status	string	Статус

2.4 Аналіз безпеки даних

Безпека даних в системі відіграє критичну роль, оскільки вона обробляє та зберігає персональні дані користувачів. Наявність надійних механізмів захисту є обов'язковими вимогами, що гарантують конфіденційність, цілісність та доступність даних. Веб- та мобільний застосунки розроблені з урахуванням суворих вимог до безпеки даних, встановлених GDPR (General Data Protection Regulation). Усі аспекти обробки персональних даних користувачів, включаючи зберігання, передачу та

доступ, піддаються ретельному аналізу з метою забезпечення повної відповідності цим нормам.

– Шифрування даних — усі персональні дані користувачів, зокрема паролі, зберігаються у зашифрованому вигляді. Firebase, який використовується для розміщення серверної частини додатка, використовує високостійкі алгоритми шифрування для забезпечення безпеки даних.

– Безпечний протокол передачі даних — усі дані, що передаються між веб-клієнтом та сервером, а також між мобільним застосунком та сервером, передаються через захищений протокол HTTPS; що гарантує шифрування даних в процесі передачі та відповідає вимогам GDPR.

– Авторизація і аутентифікація — Firebase також надає надійні сервіси авторизації та аутентифікації, що дозволяють впевнено впроваджувати контроль доступу до різних частин додатка.

– Політика конфіденційності — користувачам надається зрозуміла політика конфіденційності, яка вказує на дані що збираються та яким чином вони використовуються і зберігаються.

– Обробка помилок та відмов — у системі передбачено механізми обробки помилок та відмов, що допомагають забезпечити стабільність роботи та захист від атак, що мають на меті перенавантаження системи.

Всі ці механізми допомагають забезпечити надійний захист персональних даних користувачів, допомагають зберегти їх приватність та відповідають вимогам щодо захисту даних.

Висновки до розділу

В процесі проектування системи було здійснено детальне опрацювання архітектури застосунку, яка базується на web-first кросплатформеному підході для клієнтської частини та BaaS-архітектурі для серверної частини.

Клієнтська частина була побудована з використанням фреймворку Ionic, який у поєднанні з Capacitor дозволяє реалізовувати нативні функціональності на різних платформах, не втрачаючи при цьому гнучкості та продуктивності веб-розробки.

Серверна частина реалізована за допомогою Node.js, Express.js та Firebase Services. Використання такого поєднання технологій дозволило зосередитися на бізнес-логіці, зменшивши при цьому час і ресурси, що потребуються для підтримки інфраструктури.

Програмний код розбито на модулі за логічним принципом для полегшення розуміння структури застосунку і подальшої підтримки коду. Кожен модуль відповідає за окрему частину функціональності системи.

Було розроблено схему бази даних, що враховує всі необхідні вимоги до зберігання, обробки та доступу до даних.

Аналіз безпеки даних показав, що використання Firebase Authentication та безпеки на рівні правил бази даних Firebase забезпечує надійну захищеність даних користувачів.

У цілому, процес проектування системи дозволив визначити основні напрямки подальшої розробки, що сприяє успішній реалізації проекту.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Для оцінки якості програмного забезпечення було вирішено використовувати метрики Lighthouse і Web Vitals.

Lighthouse – це відкритий автоматизований інструмент для оцінки якості веб-сторінок за різними параметрами. Зокрема, він аналізує продуктивність, доступність, практики прогресивних веб-застосунків (PWA), SEO та інші аспекти веб-сайту. Він генерує звіт, який дає змогу побачити, які аспекти сторінки потребують вдосконалення.

Web Vitals – це ініціатива Google, що має на меті надати єдині рекомендації щодо якості користувацького досвіду в Інтернеті. Цей набір метрик вимірює відкликання, візуальну стабільність та підвантаження сторінки. Зокрема, до Web Vitals входять такі метрики як Largest Contentful Paint (LCP), First Input Delay (FID), і Cumulative Layout Shift (CLS).

Використання цих метрик дозволить забезпечити об'єктивний аналіз якості програмного забезпечення і виявити можливі вектори для його оптимізації і вдосконалення.

На рисунках 3.1 - 3.2 наведені звіти з цих інструментів:

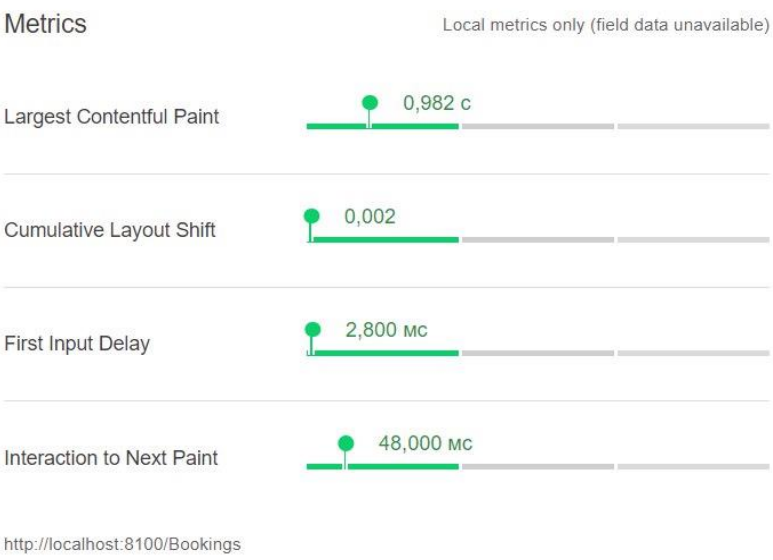


Рисунок 3.1 – Метрики Web Vitals

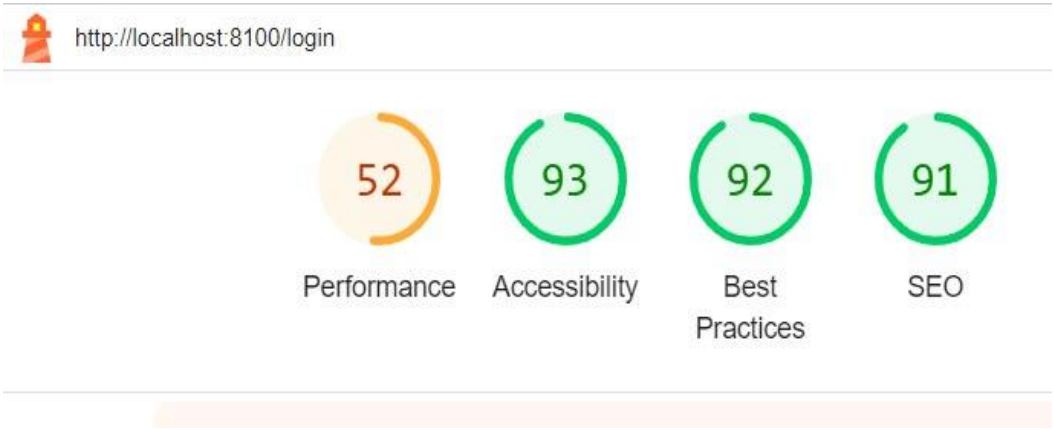


Рисунок 3.2 – Метрики Lighthouse через замірювання методом навігації

Після проведення оцінки якості нашого програмного забезпечення за допомогою метрик Lighthouse і Web Vitals, ми отримали наступні результати

Lighthouse

– Продуктивність: 52. Цей показник вказує на те, що наш застосунок працює нормально проте є можливості покращити ефективність та швидкодію.

– Доступність: 93. Це підтверджує, що наш PWA застосунок доступний для широкого кола користувачів, включно з тими, хто використовує допоміжні технології.

– SEO: 91. Оцінка показує, що наш застосунок оптимізований для пошукових систем і готовий до індексації.

– PWA: - installable. Прогресивний веб-застосунок можливо інсталювати на будь-яку платформу. Також додаток має змогу працювати в офлайн-режимі за допомогою кешування Service Worker.

Web Vitals

– LCP (Largest Contentful Paint): 0,962 мс. Це вказує на те, що основний контент на сторінках завантажується швидко.

– FID (First Input Delay): 2,8 мс. Це підтверджує, що застосунок реагує на взаємодію користувачів без затримок.

– CLS (Cumulative Layout Shift): 0,02 Низьке значення CLS свідчить про стабільність розміщення елементів на сторінках при завантаженні.

Ці результати свідчать про високу якість нашого програмного забезпечення з точки зору , доступності, SEO і користувацького досвіду. Але і про можливі покращення з точки зору продуктивності.

3.2 Опис процесів тестування

По результату написання проекту було виконане мануальне тестування функціональної частини програмного забезпечення. Опис відповідних тестів наведено у таблицях 3.1 – 3.13.

Таблиця 3.1 – Тест 1.1

Тест	Реєстрація користувача та адміністратора коворкінгу
Модуль	Реєстрація
Номер тесту	1.1
Початковий стан системи	Гість знаходиться на сторінці реєстрації
Вхідні данні	Електронна пошта, пароль, (якщо користувач то додатково поля вводу навичок)
Опис проведення тесту	Вводяться коректна електронна пошта (не зареєстрована раніше), пароль та навички для користувачів. Поля для навичок можна додавати та видаляти. Натискається кнопка реєстрації.
Очікуваний результат	Реєстрація успішна, користувач додається до системи та переходить на сторінку пошуку коворкінгів, а адміністратор на сторінку додавання інформації по коворкінгу.
Фактичний результат	Реєстрація успішна, користувач додається до системи та переходить на сторінку пошуку коворкінгів, а адміністратор на сторінку реєстрації коворкінгу.

Таблиця 3.2 – Тест 1.2

Тест	Відображення помилки реєстрації користувача
Модуль	Реєстрація користувача
Номер тесту	1.2
Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні данні	Електронна пошта, пароль, (+/- скіли)
Опис проведення тесту	Користувач вводить некоректну електронну пошту, пароль від 6 символів. Натискає кнопку реєстрації.
Очікуваний результат	Реєстрація невдала, користувач не додається у систему і відображається повідомлення про використання неправильної адреси.
Фактичний результат	Реєстрація невдала, користувач не додається у систему і відображається повідомлення про використання неправильної адреси.

Таблиця 3.3 – Тест 1.3

Тест	Авторизація користувача/ адміністратора
Модуль	Авторизація
Номер тесту	1.3
Початковий стан системи	Користувач знаходиться на сторінці авторизації
Вхідні данні	Електронна пошта, пароль
Опис проведення тесту	Користувач вводить правильна електронна пошта та пароль, який містить не менше 6 символів.
Очікуваний результат	Користувач успішно авторизується і переходить до головного екрану застосунку.
Фактичний результат	Користувач успішно авторизується і переходить до головного екрану застосунку.

Таблиця 3.4 – Тест 1.4

Тест	Введення неправильних даних для авторизації
Модуль	Авторизація користувача
Номер тесту	1.4
Початковий стан системи	Користувач знаходиться на сторінці авторизації
Вхідні данні	Неправильна електронна пошта або пароль
Опис проведення тесту	Вводиться некоректна електронна пошта або пароль.
Очікуваний результат	Авторизація користувача неуспішна, користувач бачить повідомлення з вказанням причини відмови.
Фактичний результат	Авторизація користувача неуспішна, і користувач бачить повідомлення з вказанням причини відмови.

Таблиця 3.5 – Тест 1.5

Тест	Відображення списку доступних коворкінгів
Модуль	Пошук коворкінгів
Номер тесту	1.5
Початковий стан системи	Користувач знаходиться на головному екрані пошуку коворкінгів
Вхідні данні	-
Опис проведення тесту	Перевірка, чи відображається список доступних коворкінгів на сторінці пошуку коворкінгів.
Очікуваний результат	Відображення списку доступних коворкінгів для користувача.
Фактичний результат	Відображення списку доступних коворкінгів для користувача.

Таблиця 3.6 – Тест 1.6

Тест	Зміна параметрів пошуку та відображення нових результатів
Модуль	Пошук коворкінгів
Номер тесту	1.6
Початковий стан системи	Користувач знаходиться на екрані пошуку коворкінгів
Вхідні данні	Навички користувача, локація та інтервал ціни
Опис проведення тесту	Користувач вводить нові навички (редагує старі/ видаляє), вводить нову локацію та обирає цінову категорію
Очікуваний результат	Нові параметри скілів та локації зберігаються для цього користувача в базу даних, відповідно користувач отримує новий список коворкінгів по результату обробки запиту алгоритмом
Фактичний результат	Нові параметри скілів та локації зберігаються для цього користувача в базу даних, відповідно користувач отримує новий список коворкінгів по результату обробки запиту алгоритмом

Таблиця 3.7 – Тест 1.7

Тест	Бронювання робочого місця та відображення його на сторінці бронювання
Модуль	Бронювання робочого місця
Номер тесту	1.7
Початковий стан системи	Користувач знаходиться на екрані пошуку коворкінгів, обирає коворкінг та натискає на кнопку бронювання — переходить на сторінку бронювання робочого місця.
Вхідні данні	Дата, часовий слот та робоче місце
Опис проведення тесту	Користувач на сторінці бронювання робочого місця в коворкінгу обирає дату, часовий слот та робоче місце. Натискає кнопку бронювання на відповідно обраному робочому місці. Після цього користувач на сторінці бронювань бачить нове створене бронювання.
Очікуваний результат	Нове бронювання успішно створюється та відображається на сторінці бронювань користувача.
Фактичний результат	Нове бронювання успішно створюється та відображається на сторінці бронювань користувача.

Таблиця 3.8 – Тест 1.8

Тест	Скасування актуального бронювання
Модуль	Бронювання користувача
Номер тесту	1.8
Початковий стан системи	Користувач знаходиться на сторінці своїх бронювань
Вхідні данні	-
Опис проведення тесту	Користувач натискає на кнопку скасування поруч з обраним бронюванням та натискає кнопку Confirm для підтвердження скасування.
Очікуваний результат	Статус бронювання успішно змінюється та відповідно замість кнопки скасування поруч з’являється іконка скасованого бронювання.
Фактичний результат	Статус бронювання успішно змінюється та відповідно замість кнопки скасування поруч з’являється іконка скасованого бронювання.

Таблиця 3.9 – Тест 1.9

Тест	Створення коворкінгу для зареєстрованого адміністратора
Модуль	Додавання інформації по коворкінгу
Номер тесту	1.9
Початковий стан системи	Адміністратор після реєстрації перенаправляється на сторінку додавання коворкінгу.
Вхідні данні	Назва, опис, локація, та посилання на фотографії
Опис проведення тесту	У поля для назви, опису, локації та посилань адміністратор вводить відповідні дані та потім натискає на кнопку Submit для створення коворкінгу. Після цього для адміністратора повинна бути доступна сторінка додавання робочих місць у своєму коворкінгу.
Очікуваний результат	Адміністратор перенаправляється на сторінку робочих місць в своєму коворкінгу.
Фактичний результат	Адміністратор перенаправляється на сторінку робочих місць в своєму коворкінгу.

Таблиця 3.10 – Тест 1.10

Тест	Створення робочого місця в коворкінгу
Модуль	Робочі місця
Номер тесту	1.10
Початковий стан системи	Адміністратор знаходиться на сторінці робочих місць
Вхідні данні	Позиція, кількість місць та ціна за погодинну оренду
Опис проведення тесту	Адміністратор натискає на кнопку Add place створюється картка робочого місця, де користувач вводить відповідні вхідні дані після чого натискає на кнопку-іконку галочки.
Очікуваний результат	Створюється нове робоче місце у коворкінгу в БД та відображається як картка на сторінці робочих місць.
Фактичний результат	Створюється нове робоче місце у коворкінгу в БД та відображається як картка на сторінці робочих місць.

Таблиця 3.11 – Тест 1.11

Тест	Задання доступного для бронювання часового слоту по виділеному/виділеним робочу місцю
Модуль	Робочі місця
Номер тесту	1.11
Початковий стан системи	Адміністратор знаходиться на сторінці робочих місць
Вхідні данні	Робочі місця, дата, початок часового слоту та кінець.
Опис проведення тесту	Адміністратор виділяє робоче місце або декілька та натискає на кнопку Set time, після чого обирає дату та початок і кінець часового слоту. Виходить з модального вікна шляхом натискання на кнопку Confirm.
Очікуваний результат	Адміністратор по натисканню на кнопку перегортання картки робочого місця бачить відповідний доступний часовий слот на відповідну дату.
Фактичний результат	Адміністратор по натисканню на кнопку перегортання картки робочого місця бачить відповідний доступний часовий слот на відповідну дату.

Таблиця 3.12 – Тест 1.12

Тест	Створення бронювання виділеного(виділених) робочого місця
Модуль	Робочі місця
Номер тесту	1.12
Початковий стан системи	Адміністратор знаходиться на сторінці робочих місць
Вхідні данні	Робочі місця, дата, початок часового слоту та кінець. Також опція Booked у модальному вікні.
Опис проведення тесту	Адміністратор виділяє робоче місце або декілька та натискає на кнопку Set time, після чого обирає дату та початок і кінець часового слоту, також перемикає радіо-кнопку у стан Booked. Виходить з модального вікна шляхом натискання на кнопку Confirm.
Очікуваний результат	Адміністратор по натисканню на кнопку перегортання картки робочого місця бачить відповідно створене бронювання.
Фактичний результат	Адміністратор по натисканню на кнопку перегортання картки робочого місця бачить відповідно створене бронювання.

Таблиця 3.13 – Тест 1.13

Тест	Скасування бронювання робочого місця адміністратором
Модуль	Робочі місця
Номер тесту	1.13
Початковий стан системи	Адміністратор знаходиться на сторінці робочих місць
Вхідні данні	-
Опис проведення тесту	Адміністратор натискає на кнопку перегортання картки робочого місця, бачить створені бронювання і користувачами і власні. Адміністратор натискає на кнопку скасування бронювання. Статус бронювання змінюється та замість кнопки скасування відображається іконка скасованого бронювання.
Очікуваний результат	Статус бронювання змінюється та замість кнопки скасування відображається іконка скасованого бронювання.
Фактичний результат	Статус бронювання змінюється та замість кнопки скасування відображається іконка скасованого бронювання.

3.3 Опис контрольного прикладу

Проведемо повний опис контрольного прикладу з ілюстраціями мануальних тестових прикладів на Android платформі.

Запускаємо застосунок та потрапляємо на екран авторизації/реєстрації. Протестуємо авторизацію з неправильним паролем, отримаємо помилку з вказанням її причини на рисунку 3.31

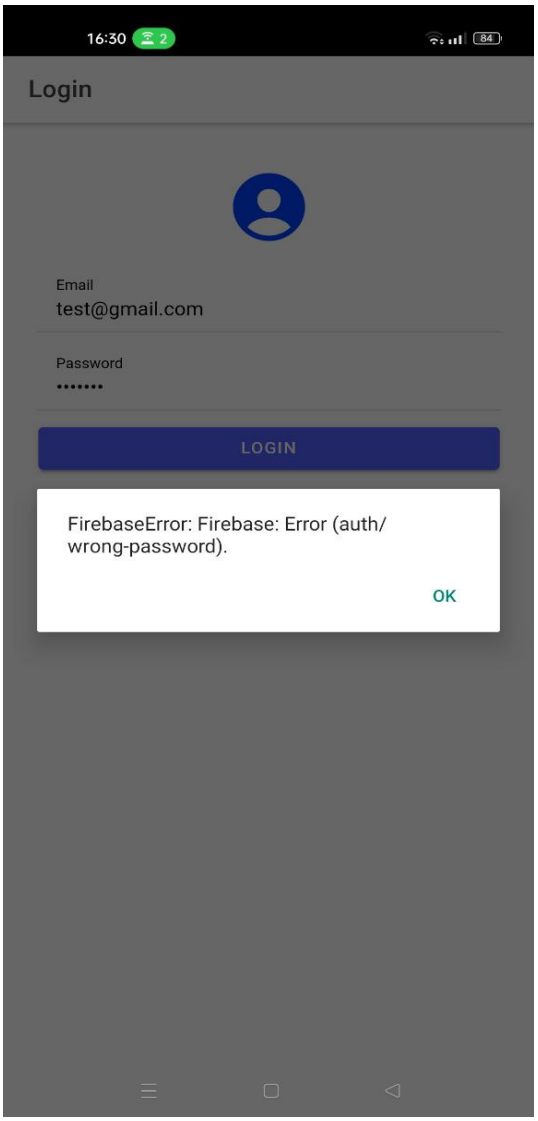


Рисунок 3.31 — помилка при авторизації (неправильно введений пароль)

Наступним кроком після реєстрації/ авторизації розглянемо флоу (шлях по сторінкам) для зареєстрованого адміністратора. Спочатку від

адміністратора потребується вказати інформацію по своєму коворкінгу — сторінку додавання коворкінгу з введеними даними вказано на рисунку 3.32

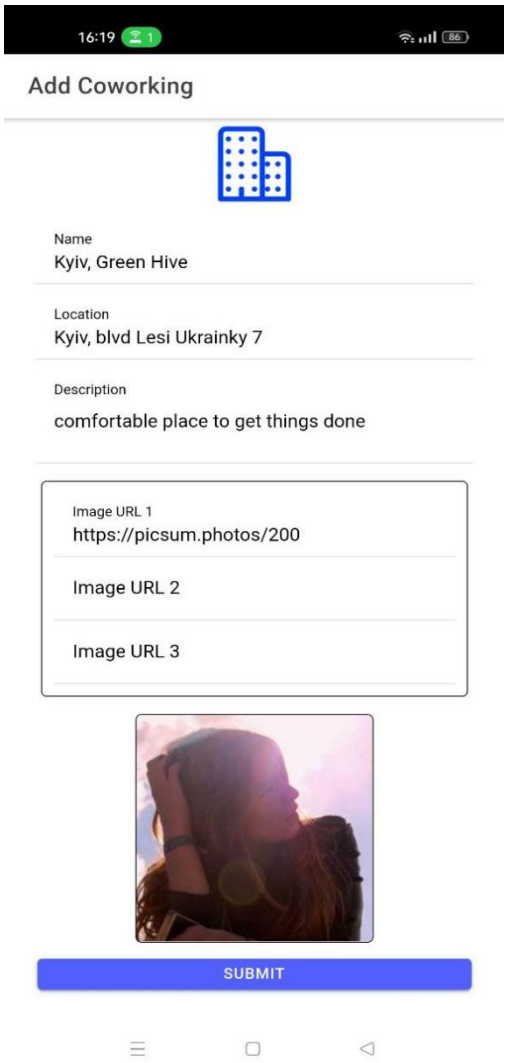


Рисунок 3.32 — реєстрація коворкінгу з введеними даними

Після введення даних коворкінгу адміністратор натискає на кнопку Submit щоб підтвердити дані та зареєструвати коворкінг — перехід на сторінку робочих місць в своєму коворкінгу. При невідповідності даних поля будуть підсвічуватись красним кольором та натиснути кнопку Submit не вдасться. Також наступного разу коли адміністратор буде авторизуватись він одразу перейде на сторінку робочих місць.

Розглянемо приклад додавання робочого місця в коворкінг, адміністратор натискає на кнопку Add Place — відображається нова картка

робочого місця, де від адміністратора очікується ввід інформації, приклад на рисунку 3.33.

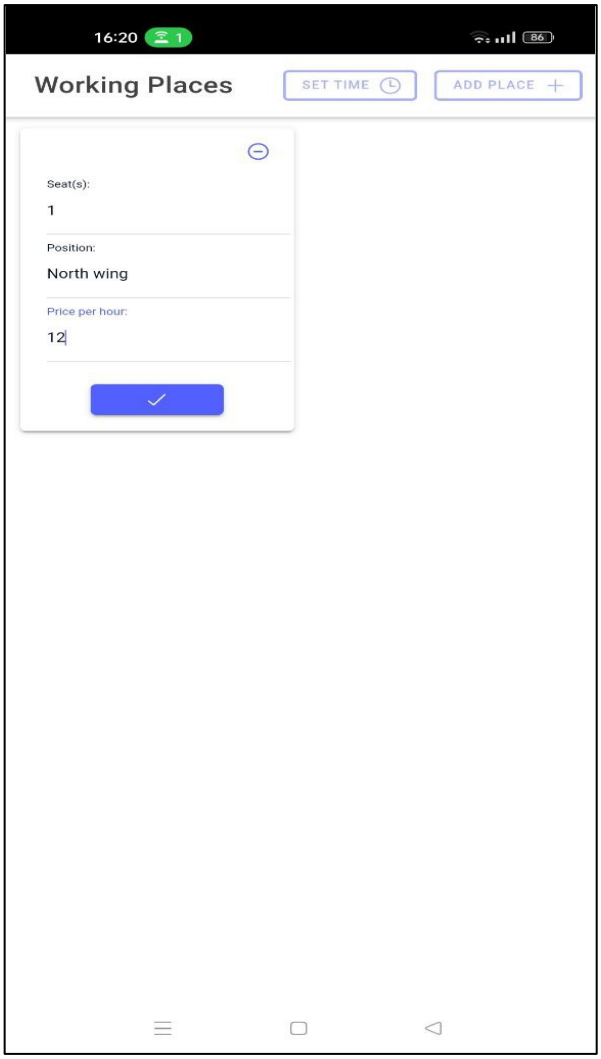


Рисунок 3.33 — додавання нового робочого місця та вказання інформації

Після вказання інформації в відповідні поля картки робочого місця користувач натискає на кнопку галочки та робоче місце зберігається. Таким чином адміністратор може додати скільки завгодно робочих місць своєму коворкінгу. Потім для того щоб зробити ці місця доступними для бронювання користувачами застосунку — адміністратору потрібно виділити робочі місця та натиснути на кнопку Set Time по натисканню на цю кнопку у адміністратора буде дві опції функціоналу у відкритому модальному вікні:

– вказати в системі стороннє бронювання робочого місця (місць);
– задати місцю (місцям) доступні часові слоти для бронювання.

Таким чином адміністратор додає свій коворкінг з робочими місцями доступними для бронювання у алгоритм підбору коворкінгу для користувачів. З точки зору флоу для користувача наступною сторінкою після реєстрації/ авторизації буде сторінка пошуку коворкінгу де користувач зможе обрати свій варіант коворкінгу та налаштувати (персоналізувати) алгоритм для відображення інших результатів. На рисунках 3.34 і 3.35 наведено приклади сторінки пошуку коворкінгу та модального вікна налаштування підбору.

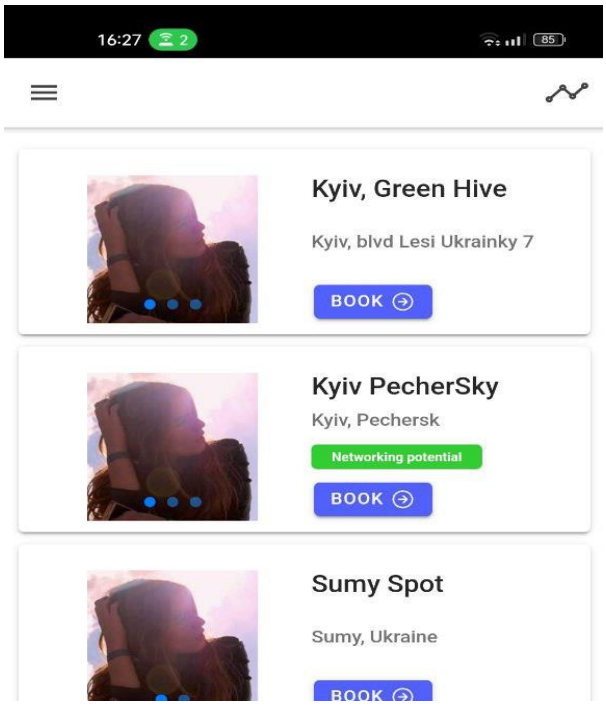


Рисунок 3.34 — сторінка пошуку коворкінгів

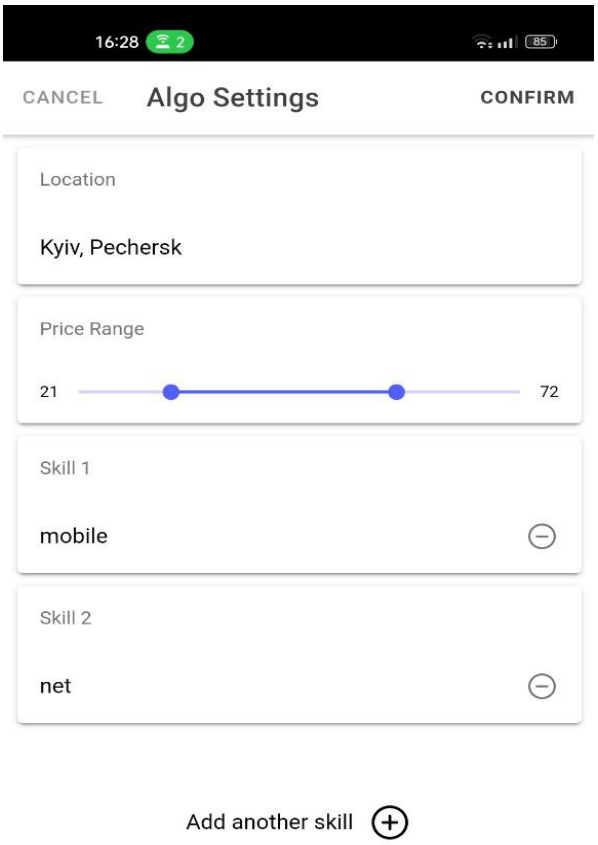


Рисунок 3.35 — модальне вікно налаштування пошуку коворкінгів

Коли користувач обирає коворкінг та хоче забронювати місце у ньому він натискає на кнопку Book та перенаправляється на сторінку бронювання місць. Там користувач бачить детальну інформацію по коворкінгу та обирає дату із доступних для бронювання, по цій даті доступні часові слоти — та відповідні доступні робочі місця на ці час і дату.

Далі на рисунку 3.36 наведено приклад сторінки бронювання робочих місць коворкінгу, після натискання на кнопку Book обраного робочого місця користувачу потрібно буде підтвердити своє бронювання. Користувач матиме змогу переглянути усі свої бронювання на окремій сторінці, а також скасувати свої актуальні бронювання — приклад наведено на рисунку 3.37

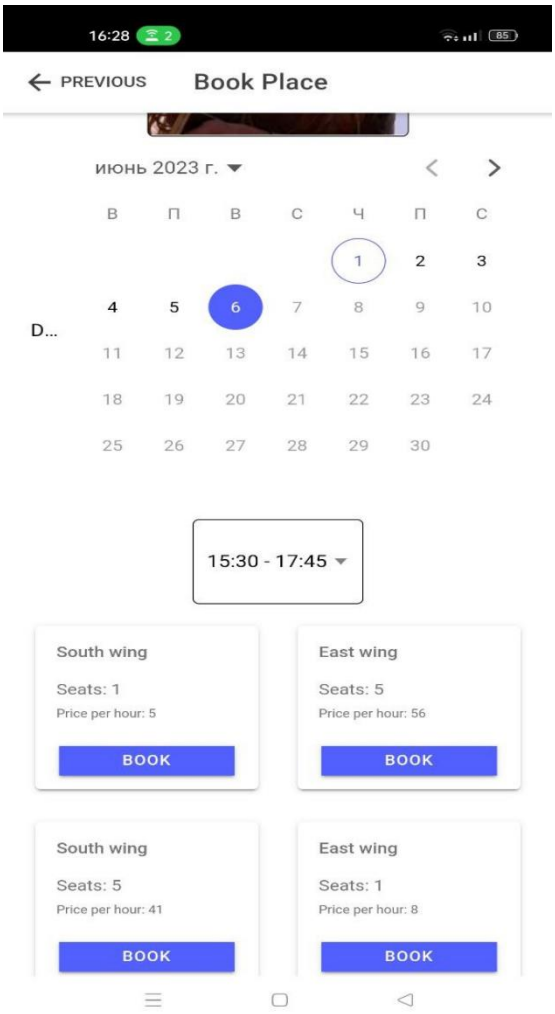


Рисунок 3.36 — сторінка бронювання робочого місця в коворкінгу з обраною датою та відповідно доступним часовим слотом

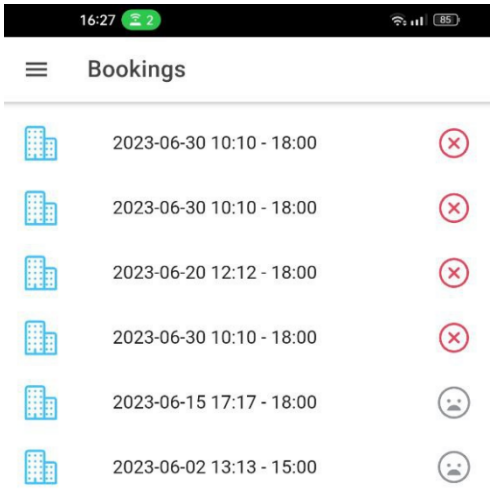


Рисунок 3.37 — сторінка усіх бронювань користувача (два останні скасовані, 4 перших актуальні бронювання робочих місць)

На сторінці усіх бронювань користувач може скасовувати актуальні бронювання, переглядати усю історію своїх бронювань та за натисканням на кнопку коворкінгу зліва може перейти на сторінку бронювання робочих місць коворкінгу та переглянути детальну інформацію про коворкінг.

Висновки до розділу

В ході аналізу якості ПЗ було використано метрики Lighthouse і Webvitals. Результати аналізу показали високий рівень якості розробленого програмного забезпечення: продукт відповідає ключовим вимогам до швидкості завантаження, доступності, використання сучасних технологій та оптимізації для мобільних пристроїв.

Процеси тестування були детально описані і включали в себе як автоматизоване тестування, так і ручне. Автоматизоване тестування було здійснено з використанням єдиного набору інструментів, що дозволило забезпечити стабільність та безпеку застосунку. Ручне тестування, в свою чергу, дозволило виявити та виправити помилки, які могли б сховатись від автоматизованих методів.

Щодо контрольного прикладу, було продемонстровано реальне використання програмного забезпечення, що наочно ілюструвало всі його можливості та переваги. Цей приклад підтверджує ефективність і практичність розробленого ПЗ для управління та моніторингу роботи коворкінгів.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Процес розгортання програмного забезпечення можна поділити на три основні частини: розгортання серверної частини через Firebase Functions, розгортання веб-застосунку через Firebase Hosting, та розгортання Android-версії застосунку через Ionic Capacitor і ApkFab.

Розгортання серверної частини через Firebase Functions:

– ініціалізація Firebase Functions у проекті за допомогою команди firebase init functions;

– написання коду для обробки вхідних запитів та оновлення статусу бронювань;

– деплой коду за допомогою команди firebase deploy --only functions;

– перевірка працездатності обробника запитів та функції оновлення статусу бронюванню.

Розгортання PWA веб-застосунку через Firebase Hosting:

– ініціалізація Firebase Hosting у проекті за допомогою команди firebase init hosting;

– виконання команди npm run build для створення оптимізованої продакшн версії веб-застосунку;

– деплой PWA до Firebase Hosting за допомогою команди firebase deploy --only hosting;

– перевірка доступності та функціональності застосунку;

Розгортання Android-версії застосунку через Ionic Capacitor і ApkFab:

– виконання команди ionic build для створення веб-версії застосунку;

– ініціалізація Ionic Capacitor для Android за допомогою команди npx cap sync android;

– компіляція Android-версії застосунку за допомогою команди npx cap sync android і в Android Studio і збірка APK файлу застосунку;

– завантаження APK файлу до ApkFab, заповнення необхідної інформації та публікація застосунку.

4.2 Підтримка програмного забезпечення

Підтримка програмного забезпечення є важливою складовою його життєвого циклу і включає в себе ряд дій, спрямованих на забезпечення його ефективної роботи та вдосконалення. Вона виконується в декілька етапів:

– моніторинг і виправлення помилок — включає в себе моніторинг системи на наявність будь-яких вад або помилок, а також їх виправлення. Для цього можна використовувати систему відстеження помилок Firebase Crashlytics, яка дозволяє виявляти помилки в режимі реального часу та надає детальну інформацію про них;

– оновлення системи — включає в себе регулярне оновлення програмного забезпечення з метою поліпшення його функціональності та продуктивності. Оновлення можуть включати в себе внесення нових функцій, виправлення помилок або покращення виконання;

– технічна підтримка користувачів — включає в себе надання допомоги користувачам у випадку виникнення будь-яких проблем з програмним забезпеченням, технічних проблем, керівництво користувачів через процес використання програми, або допомога в вирішенні будь-яких інших проблем, що виникають під час використання програмного забезпечення;

– забезпечення безпеки — в рамках підтримки ПЗ також здійснюється постійний моніторинг на предмет забезпечення безпеки. Це включає в себе перевірку на наявність потенційних уразливостей і регулярне оновлення системи, щоб вона була захищена від нових загроз;

– вдосконалення продукту — на основі відгуків користувачів та аналізу їх поведінки пропонуються і впроваджуються удосконалення програмного забезпечення. Важливо постійно розуміти потреби користувачів і адаптувати продукт відповідно до цих потреб.

Отже, підтримка програмного забезпечення — це постійний процес, який вимагає регулярного моніторингу, аналізу і вдосконалення, щоб забезпечити найкращий досвід користувача і продуктивну роботу програмного забезпечення.

Ionic Automations і Firebase надають потужні інструменти для підтримки та оновлення програмного забезпечення.

Ionic Automations — це централізований пайплайн автоматизації, який дозволяє автоматично тестувати, будувати і розгортати застосунки. Він включає в себе ряд вбудованих автоматичних задач, які дозволяють виконувати різні дії, такі як компіляція коду, запуск тестів і деплой застосунку.

У контексті підтримки програмного забезпечення Ionic Automations може використовуватись для:

– автоматичного запуску юніт тестів та тестів UI/UX для перевірки якості коду та його відповідності очікуваному поводженню;

– автоматичного деплою оновлень коду через різні середовища (тестові, стейджинг, продакшн), що підвищує швидкість внесення оновлень та зменшує можливість помилок при ручному деплої;

– Firebase також пропонує ряд інструментів для підтримки та оновлення застосунків:

– Firebase Remote Config сервіс дозволяє динамічно оновлювати параметри та конфігурацію застосунку без необхідності випускати нову

версію. Це може бути особливо корисно при внесенні маленьких оновлень та тестування нових функцій;

– Firebase Crashlytics потужний інструмент для відстеження та аналізу помилок, що допомагає виявляти та виправляти проблеми зі стабільністю застосунку;

– Firebase Functions сервіс, що дозволяє виконувати серверний код на інфраструктурі Google Cloud в режимі реального часу. Це дуже зручно для внесення оновлень на серверну застосунку;

Таким чином, за допомогою Ionic Automations та Firebase, ми автоматизуємо багато процесів, пов'язаних з підтримкою та оновленням програмного забезпечення, забезпечуючи його безперервну роботу та вдосконалення.

Висновки до розділу

Із дослідження розгортання програмного забезпечення випливає, що використання Firebase Hosting для веб-застосунків, Firebase Function для серверної логіки та Ionic Capacitor для створення нативних мобільних додатків є ефективним та надійним підходом. Ці інструменти спрощують процес деплою, забезпечуючи одночасно високу продуктивність та стабільність застосунку. Крім того, завдяки автоматизації процесу розгортання, зменшується можливість виникнення помилок.

Ionic Automations та Firebase дозволяє автоматизувати важливі процеси підтримки, такі як виявлення та виправлення помилок, проведення тестів, оновлення конфігурації та деплоєм оновлень. Це не лише полегшує процес підтримки, але й сприяє покращенню якості програмного продукту, його стабільності та продуктивності.

Додатково, можна відмітити, що централізована система підтримки на основі Firebase значно спрощує процес моніторингу застосунку, надаючи

засоби для візуалізації та аналізу використання ресурсів, виявлення помилок та відстеження дій користувачів.

З іншого боку, прив'язка процесів розгортання та підтримки до Ionic Automations сприяє забезпеченню безперебійного циклу розробки та впровадження. Завдяки автоматичному оновленню, розробникам не потрібно вручну виконувати складні операції з деплоєм кожного оновлення, що зменшує ризик внесення помилок та підвищує швидкість впровадження нових функцій та виправлень.

Отже, розглянуті методи розгортання та підтримки програмного забезпечення виявилися ефективними, адаптованими до поточних потреб та спрямованими на високу продуктивність та якість кінцевого продукту.

ВИСНОВКИ

В результаті виконання дипломного проєкту було спроєктовано програмне забезпечення для управління та моніторингу роботи коворкінгів. Для розробки використовувались комбінація різних програмних засобів, включаючи Firebase, Node для серверної частини, Ionic, React для клієнтської частини.

Найважливіші наукові й практичні результати роботи включають:

– розроблення рекомендаційного алгоритму для ефективного моніторингу коворкінгів, враховуючи динамічні потреби користувачів і бізнес-процеси;

– успішне створення прототипу програмного забезпечення, його тестування та вдосконалення на основі зворотного зв'язку користувачів.

Ступінь впровадження та можливі галузі або сфери використання результатів роботи: розроблена система може бути використана у сфері управління коворкінгами. Впровадження системи сприятиме зниженню часу та зусиль, необхідних для управління ресурсами та моніторингу стану коворкінгу, що покращить продуктивність роботи менеджерів.

Наукова, науково-технічна, соціально-економічна значущість роботи: розроблені алгоритми моніторингу та управління на основі аналізу потреб користувачів та бізнес-процесів мають наукову значущість у вдосконаленні процесу управління коворкінгами. Застосування програмного забезпечення може мати соціально-економічний вплив, забезпечуючи більш ефективне управління ресурсами та скорочуючи час, необхідний для управління коворкінгом.

Модифіковано:

– було модифіковано алгоритми моніторингу та управління з метою забезпечення кращої точності та збалансованості відповідно до потреб коворкінгу та користувачів;

– було вдосконалено параметри пошуку і моніторингу коворкінгів, враховуючи додаткові фактори, такі як динаміка користування простором, зміни в потребах користувачів та ін.

Стан вирішення усіх поставлених в дипломному проєктуванні задач: усі поставлені в дипломному проєктуванні задачі були успішно вирішені. Було проведено аналіз потреб коворкінгів, розроблено алгоритм рекомендаційного моніторингу, створено прототип програмного забезпечення, проведено його тестування з участю користувачів та вдосконалено на основі отриманого зворотного зв'язку. Результати тестування та аналізу ефективності підтвердили успішну реалізацію програмного забезпечення для управління та моніторингу роботи коворкінгів.

Отже, дипломна робота є актуальною, а результати її виконання можуть бути успішно впроваджені у сфері управління коворкінгами. Продовження досліджень у даній тематиці є доцільним з метою подальшого вдосконалення алгоритмів моніторингу та управління, розширення функціональності програмного забезпечення та його адаптації для інших сфер застосування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Brown J. / Curating the “Third Place”? Coworking and the mediation of creativity / J. Brown // Geoforum. – 2017. – Vol. 82. – P. 112–126.

2) Bouncken R.B., Reuschl A.J. / Coworking-spaces: how a phenomenon of the sharing economy builds a novel trend for the workplace and for entrepreneurship / R.B. Bouncken, A.J. Reuschl // Review of Managerial Science. – 2018. – Vol. 12. – P. 317–334.

3) Spreitzer G., Bacevice P., Garrett L. / Why People Thrive in Coworking Spaces [Electronic resource] / G. Spreitzer, P. Bacevice, L. Garrett // Harvard Business Review. – 2015. – Mode of access: https://hbr.org/2015/05/why-people-thrive-in-coworking-spaces

4) Richter A. Reassembling austerity research / A. Richter, H. Hilbrandt // Coworking in the City / A. Richter, H. Hilbrandt., 2015. – Vol. 15(1). – P. 163–180.

5) Firebase Documentation [Electronic resource]. – Mode of access: https://firebase.google.com/docs

6) Ionic Documentation [Electronic resource]. – Mode of access: https://ionicframework.com/docs

7) Capacitor Documentation [Electronic resource]. – Mode of access: https://capacitorjs.com/docs

8) React Documentation [Electronic resource]. – Mode of access: https://react.dev

9) Visual Studio Documentation [Electronic resource]. – Mode of access: https://code.visualstudio.com/docs

10) Android Studio Documentation [Electronic resource]. – Mode of access: https://developer.android.com/docs

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015398488

Дата перевірки:
02.06.2023 16:33:40 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
02.06.2023 16:35:56 EEST

ID користувача:
76913

Назва документа: **IT-92_Русановський_ПЗ**

Кількість сторінок: **79** Кількість слів: **10493** Кількість символів: **84411** Розмір файлу: **2.57 MB** ID файлу: **1015062492**

9.86%
Схожість

Найбільша схожість: **4.93%** з джерелом з Бібліотеки (ID файлу: **1015058518**)

2.97% Джерела з Інтернету	151	Сторінка 81
9.58% Джерела з Бібліотеки	187	Сторінка 83

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ ТА МОНІТОРИНГУ РОБОТИ КОВОРКІНГІВ

Текст програми

КПІ.ІТ-9221.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Ірина ВІТКОВСЬКА

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Дмитро РУСАНОВСЬКИЙ

Київ – 2023

Файл App.tsx

```
import React, { useState, useEffect } from 'react';
import { Redirect, Route } from 'react-router-dom';
import {
  IonApp,
  IonPage,
  IonRouterOutlet,
  IonSplitPane,
  setupIonicReact,
} from '@ionic/react';
import { IonReactRouter } from '@ionic/react-router';
import { register } from 'swiper/element/bundle';
import Menu from './components/Menu';
import Login from './pages/Login';
import Register from './pages/Register';
import AddCoworking from './pages/AddCoworking';
import WorkingPlaces from './pages/WorkingPlaces';
import Bookings from './pages/Bookings';
import Coworkings from './pages/Coworkings';
import CoworkingPage from './pages/CoworkingPage';

/* Core CSS required for Ionic components to work properly */
import '@ionic/react/css/core.css';

/* Basic CSS for apps built with Ionic */
import '@ionic/react/css/normalize.css';
import '@ionic/react/css/structure.css';
import '@ionic/react/css/typography.css';

/* Optional CSS utils that can be commented out */
import '@ionic/react/css/padding.css';
import '@ionic/react/css/float-elements.css';
import '@ionic/react/css/text-alignment.css';
import '@ionic/react/css/text-transformation.css';
import '@ionic/react/css/flex-utils.css';
import '@ionic/react/css/display.css';

/* Theme variables */
import './theme/variables.css';

import {
  auth,
```

```
  getAuth,
  db,
  doc,
  getDoc,
  onAuthStateChanged,
} from './firebase/config';
import { getCoworkingId } from './firebase/functions';
import { DocumentData } from 'firebase/firestore';
import { User } from './types';

setupIonicReact({
  rippleEffect: false,
  scrollAssist: false,
});

// register Swiper custom elements
register();

const App: React.FC = () => {
  const [loading, setLoading] = useState(true);
  const [user, setUser] = useState<DocumentData | undefined | null>(null);

  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, async (currentUser) => {
      if (currentUser) {
        const userRef = doc(db, 'users', currentUser.uid);
        const userSnapshot = await getDoc(userRef);

        // .then((document) => {
        const userData = userSnapshot.data();
        console.log(userData);
        // setLoading(false);
        setUser(userData);
        // })
        // .catch((error) => {
        // alert(error);
        setLoading(false);
        // });
      } else {
        console.error('No currentUser');
        // history.push('/login');
        setLoading(false);
      }
```

```
    });

    // Unsubscribe on cleanup
    return () => unsubscribe();
  }, [auth]);

  if (loading) {
    return <IonPage>Loading...</IonPage>;
  }

  return (
    <IonApp>
      <IonReactRouter>
        <IonSplitPane contentId='main'>
          <Menu
            disabled={!(user && !user.admin)}
            email={user ? user.email : ''}
          />
          <IonRouterOutlet id='main'>
            <Route
              path='/'
              exact
              render={(props) => <Redirect to='/login' />}
            />
            <Route
              path='/login'
              exact
              render={(props) => <Login {...props} />}
            />
            <Route
              path='/register'
              exact
              render={(props) => <Register {...props} />}
            />
            <Route
              path='/Bookings'
              exact
              render={(props) =>
                user && !user.admin ? (
                  <Bookings user={user} {...props} />
                ) : (
                  <Redirect to='/login' />
                )
```

```
    }
  />
  <Route
    path='/Coworkings'
    exact
    render={(props) =>
      user ? (
        <Coworkings user={user} {...props} />
      ) : (
        <Redirect to='/login' />
      )
    }
  />
  <Route
    path='/Coworkings/:id'
    exact
    render={(props) =>
      user ? (
        <CoworkingPage user={user} {...props} />
      ) : (
        <Redirect to='/login' />
      )
    }
  />

  <Route
    path='/addCoworking'
    exact
    render={(props) =>
      user && user.admin ? (
        <AddCoworking user={user} {...props} />
      ) : (
        <Redirect to='/login' />
      )
    }
  />

  <Route
    path='/WorkingPlaces'
    exact
    render={(props) =>
      user && user.admin ? (
        <WorkingPlaces user={user} {...props} />
```

```
                ) : (
                  <Redirect to='/login' />
                )
              }
            />
          </IonRouterOutlet>
        </IonSplitPane>
      </IonReactRouter>
    </IonApp>
  );
};

export default App;
```

Файл types/index.ts

```
import { DocumentData, Timestamp } from "firebase/firestore";

export type TimeSlot = {
  startTime: { hour: string, minute: string };
  endTime: { hour: string, minute: string };
};

export type AvailableDate = {
  date: string;
  timeSlots: TimeSlot[];
};

export type NewAvailableDatesTime = {
  dates: string[];
  timeSlot: TimeSlot;
};

export type WorkingPlace = {
  id: string;
  seats: number;
  position: string;
  pricePerHour: number;
  createdAt?: Timestamp;
  availableDates: { [date: string]: TimeSlot[] };
};
```

```
export type Booking = {
  id: string;
  createdAt?: Timestamp;
  userId: string;
  workingPlaceId: string;
  coworkingId: string;
  status?: 'active' | 'completed' | 'canceled';
  date?: string;
  timeSlot: TimeSlot;
}

export type WithRequired<T, K extends keyof T> = T & { [P in K]-?: T[P] };

export type CreateBookingData = Omit<Booking, 'id' | 'createdAt' | 'status'>;

export type User = {
  id: string;
  email: string;
  userSkills?: string[];
  admin?: boolean;
};

export type CoworkingItemType = {
  id: string;
  name: string;
  location: string;
  imageUrls: string[];
  description: string;
  networking?: boolean;
};

export type RangeValue = {
  lower: number;
  upper: number;
};

export type newSettings = {
  location: string;
  skills: string[];
  priceRange: RangeValue;
};

export type CustomizedState = { coworkingId: string };
```

```
export interface DocumentDataInterface {
  user: DocumentData;
}
```

Файл AddCoworking.tsx

```
import {
  IonContent,
  IonHeader,
  IonPage,
  IonTitle,
  IonToolbar,
  IonGrid,
  IonRow,
  IonCol,
  IonItem,
  IonLabel,
  IonInput,
  IonButton,
  IonIcon,
  IonAlert,
  IonTextarea,
} from '@ionic/react';
import React, { useState } from 'react';
import { businessOutline } from 'ionicons/icons';
import { RouteComponentProps } from 'react-router';
import { Swiper, SwiperSlide } from 'swiper/react';
import { Navigation, Pagination } from 'swiper';
import 'swiper/css/navigation';
import 'swiper/css/pagination';
import {
  addDoc,
  collection,
  db,
  doc,
  serverTimestamp,
  setDoc,
} from '../firebase/config';
import { CustomizedState } from '../types';
import { DocumentData } from 'firebase/firestore';

interface BookingsPageProps extends RouteComponentProps {
```

```
  user: DocumentData;
}
const AddCoworking: React.FC<BookingsPageProps> = ({ user, history }) => {
  const [name, setName] = useState<string>('');
  const [location, setLocation] = useState<string>('');
  const [description, setDescription] = useState<string>('');
  const [imageUrls, setImageUrls] = useState<string[]>(['', '', '']);
  const [iserror, setIserror] = useState<boolean>(false);
  const [message, setMessage] = useState<string>('');

  const handleImageUrl = (url: string, index: number) => {
    setImageUrls((prevUrls) => {
      const newUrls = [...prevUrls];
      newUrls[index] = url;
      return newUrls;
    });
  };

  const handleSubmit = async () => {
    if (!name) {
      // setMessage('Please enter the name of your coworking');
      // setIserror(true);
      // return;
    }

    if (!location) {
      // setMessage('Please enter location');
      // setIserror(true);
      // return;
    }

    // if (description.length < 30) {
    //   setMessage('Description must be at least 30 characters');
    //   setIserror(true);
    //   return;
    // }

    // if (imageUrls.filter((url) => url).length === 0) {
    //   setMessage('Please provide at least one image URL');
    //   setIserror(true);
    //   return;
    // }
```

```
  // try {
  // const coworkingRef = collection(db, 'coworkings');
  // const timestamp = serverTimestamp();
  //   coworkings.map(async (coworking) => {
  //     const data = {
  //       name: coworking.name,
  //       location: coworking.location,
  //       description: coworking.description,
  //       imageUrls: coworking.imageUrls.concat(coworking.imageUrls),
  //       authorID: (window.history as any).state.state.user.id,
  //       createdAt: timestamp,
  //     };
  //     await addDoc(coworkingRef, data);
  //   });
  try {
    const coworkingRef = collection(db, 'coworkings');
    const timestamp = serverTimestamp();
    const data = {
      name,
      location,
      description,
      imageUrls,
      authorID: user.id,
      createdAt: timestamp,
    };
    // if (!id) {
    await addDoc(coworkingRef, data);
    // } else {
    // await setDoc(doc(coworkingRef, id), data);
    // }
    history.push('/WorkingPlaces', { coworkingId: coworkingRef.id });
  } catch (error) {
    console.log(error);
  }
};

return (
  <IonPage>
    <IonHeader>
      <IonToolbar>
        <IonTitle>Add Coworking</IonTitle>
      </IonToolbar>
    </IonHeader>
```

```
<IonContent fullscreen className='ion-padding ion-text-center'>
  <IonGrid>
    <IonRow>
      <IonCol>
        <IonAlert
          isOpen={iserror}
          onDidDismiss={() => setIserror(false)}
          cssClass='my-custom-class'
          header={'Error!'}
          message={message}
          buttons={['Dismiss']}
        />
      </IonCol>
    </IonRow>
    <IonRow>
      <IonCol>
        <IonIcon
          style={{ fontSize: '70px', color: '#0040ff' }}
          icon={businessOutline}
        />
      </IonCol>
    </IonRow>
    <IonRow>
      <IonCol>
        <IonItem>
          <IonLabel position='floating'>Name</IonLabel>
          <IonInput
            type='text'
            value={name}
            onIonInput={(e) => setName(e.detail.value!)}
          ></IonInput>
        </IonItem>
      </IonCol>
    </IonRow>

    <IonRow>
      <IonCol>
        <IonItem>
          <IonLabel position='floating'>Location</IonLabel>
          <IonInput
            type='text'
            value={location}
            onIonInput={(e) => setLocation(e.detail.value!)}
```

```
              ></IonInput>
            </IonItem>
          </IonCol>
        </IonRow>

        <IonRow>
          <IonCol>
            <IonItem>
              <IonLabel position='floating'>Description</IonLabel>
              <IonTextarea
                value={description}
                onIonInput={(e) => setDescription(e.detail.value!)}
              ></IonTextarea>
            </IonItem>
          </IonCol>
        </IonRow>

        <IonRow>
          <IonCol
            style={{
              border: '1px solid #000',
              borderRadius: '5px',
              padding: '10px',
              margin: '10px',
            }}
          >
            {imageUrls.map((url, index) => (
              <IonItem key={index}>
                <IonLabel position='floating'>Image URL {index +
1}</IonLabel>
                <IonInput
                  type='url'
                  value={url}
                  onIonInput={(e) =>
                    handleImageUrl(e.detail.value || '', index)
                  }
                ></IonInput>
              </IonItem>
            ))}
          </IonCol>
        </IonRow>

        <IonRow>
```

```
            <IonCol>
              <div
                style={{
                  width: '200px',
                  height: '200px',
                  border: '1px solid #000',
                  borderRadius: '5px',
                  overflow: 'hidden',
                  margin: 'auto',
                }}
              >
                <Swiper
                  modules={[Pagination]}
                  pagination={{ clickable: true }}
                  slidesPerView={1}
                  spaceBetween={10}
                >
                  {imageUrls
                    .filter((url) => url)
                    .map((url, index) => (
                      <SwiperSlide key={index}>
                        <img
                          src={url}
                          alt={`Slide ${index}`}
                          style={{ width: '100%', height: '100%' }}
                        />
                      </SwiperSlide>
                    ))}
                </Swiper>
              </div>
            </IonCol>
          </IonRow>

          <IonRow>
            <IonCol>
              <IonButton size='small' expand='block' onClick={handleSubmit}>
                Submit
              </IonButton>
            </IonCol>
          </IonRow>
        </IonGrid>
      </IonContent>
    </IonPage>
```

```
  );
};

export default AddCoworking;
```

Файл Bookings.tsx

```
import React, { useEffect, useState } from 'react';
import {
  IonContent,
  IonHeader,
  IonPage,
  IonTitle,
  IonToolbar,
  IonList,
  IonItem,
  IonLabel,
  IonButton,
  IonIcon,
  IonButtons,
  IonMenuButton,
  IonAlert,
} from '@ionic/react';
import { checkmarkDoneCircleOutline, closeCircleOutline, sadOutline,
businessOutline } from 'ionicons/icons';
import { Booking, CustomizedState, DocumentDataInterface } from '../types';
import { bookingsMock } from '../components/WorkingPlaceCard';
import { cancelBooking, getBookingsByUserId } from '../firebase/functions';
import { DocumentData } from 'firebase/firestore';
import { RouteComponentProps } from 'react-router';

interface BookingsPageProps extends RouteComponentProps {
  user: DocumentData;
}

const Bookings: React.FC<BookingsPageProps> = ({ user, history, location,
match }) => {
  const [showDeleteAlert, setShowDeleteAlert] = useState<[boolean,
string]>([false, '']);
  const [bookings, setBookings] = useState<Booking[]>(bookingsMock);

  useEffect(() => {
```

```
    const fetchBookings = async () => {
      const data = await getBookingsByUserId(user.id);
      if (data.length) {
        setBookings(data);
      } else {
        setBookings(bookingsMock);
      }
    };

    fetchBookings();
  }, [user.id]);

  const handleMoveToCoworkingPage = async (coworkingId: string) => {
    if (coworkingId.startsWith('co')) {
      console.log('not moving cause it is mocked (not existing) coworking');
    } else {
      history.push(`/Coworkings/${coworkingId}`);
    }
  };

  const handleConfirmDelete = async () => {
    console.log('Canceling booking: ', showDeleteAlert[1]);
    if (!showDeleteAlert[1].startsWith('bo')) {
      await cancelBooking(showDeleteAlert[1]);
    }
    setBookings(prevBookings =>
      prevBookings.map(booking =>
        booking.id === showDeleteAlert[1] ? { ...booking, status: 'canceled'
} : booking
      )
    );
    setShowDeleteAlert([false, '']);
  };

  return (
    <IonPage>
      <IonHeader>
        <IonToolbar>
          <IonButtons slot='start'>
            <IonMenuButton />
          </IonButtons>
          <IonTitle>Bookings</IonTitle>
        </IonToolbar>
```

```
      </IonHeader>
      <IonContent>
        <IonList>
          {bookings.map(
            (
              { id, coworkingId, date, status, timeSlot: { startTime, endTime
} }
            ) => (
              <IonItem key={id} className='ion-item-booking ion-item-label'
lines='none'>
                <IonIcon
                  slot='start'
                  size='large'
                  color='secondary'
                  icon={businessOutline}
                  onClick={() => handleMoveToCoworkingPage(coworkingId)}
                />
                <p style={{ marginLeft: '14px', marginRight: '7px', width:
'50vw' }}>{`${date} ${startTime.hour}:${startTime.hour} -
${endTime.hour}:${endTime.minute}`}</p>
                {
                  status === 'active' ?
                    <IonIcon
                      slot='end'
                      size='large'
                      color='danger'
                      icon={closeCircleOutline}
                      onClick={() => setShowDeleteAlert([true, id])}
                    /> : status === 'canceled' ?
                      <IonIcon
                        slot='end'
                        size='large'
                        color='medium'
                        icon={sadOutline}
                      /> : status === 'completed' ?
                        <IonIcon
                          slot='end'
                          size='large'
                          color='success'
                          icon={checkmarkDoneCircleOutline}
                        /> : null
                }
              </IonItem>
```

```
            ))}
          </IonList>
          <IonAlert
            isOpen={showDeleteAlert[0]}
            onDidDismiss={() => setShowDeleteAlert([false, ''])}
            header={'Confirm Cancel'}
            message={'cancel this booking?'}
            className='custom-alert'
            buttons={[
              {
                text: 'No',
                role: 'cancel',
                cssClass: 'secondary',
                handler: () => setShowDeleteAlert([false, ''])
              },
              {
                text: 'Yes',
                role: 'destructive',
                handler: handleConfirmDelete
              }
            ]}
          />
        </IonContent>
      </IonPage>
    );
};

export default Bookings;
```

Файл CoworkingPage.tsx

```
import React, { useState, useEffect } from 'react';
import { DocumentData } from 'firebase/firestore';
import {
  IonPage,
  IonContent,
  IonHeader,
  IonToolbar,
  IonButton,
  IonButtons,
  IonBackButton,
  IonTitle,
  IonGrid,
```

```
  IonRow,
  IonCol,
  IonDatetime,
  IonLabel,
  IonTextarea,
  IonItem,
  IonText,
  IonLoading,
  IonSelect,
  IonSelectOption,
  IonCard,
  IonCardContent,
  IonCardSubtitle,
  IonCardHeader,
  IonAlert,
  IonImg,
} from '@ionic/react';
import { RouteComponentProps } from 'react-router';
import { Swiper, SwiperSlide } from 'swiper/react';
import { Pagination } from 'swiper';
import 'swiper/swiper.css';
import 'swiper/css/pagination';

import { WorkingPlace, CoworkingItemType, TimeSlot } from '../types';
import { mockWorkingPlaces } from './WorkingPlaces';
import {
  getCoworkingInfoById,
  getWorkingPlaces,
  createBooking,
} from '../firebase/functions';

export const mockedAvailableDatesAndTimeSlots = [
  {
    date: '2023-06-01',
    timeSlots: [
      {
        startTime: { hour: '15', minute: '00' },
        endTime: { hour: '17', minute: '15' },
      },
      {
        startTime: { hour: '17', minute: '30' },
        endTime: { hour: '18', minute: '45' },
      },
```

```
    ],
  },
];

interface CoworkingPageProps extends RouteComponentProps<{ id: string }> {
  user: DocumentData;
}

const CoworkingPage: React.FC<CoworkingPageProps> = ({
  user,
  match,
  location,
  history,
}) => {
  // const { name, location: coworkingLocation, description, imageUrls,
networking } = location.state as CoworkingItemType;
  // console.log({ name, location: coworkingLocation, description, imageUrls,
networking });

  const [coworking, setCoworking] = useState<CoworkingItemType | null>(null);
  const [workingPlaces, setWorkingPlaces] = useState<WorkingPlace[] |
[]>([]);
  const [bookingPlaceId, setBookingPlaceId] = useState<string | null>(null);
  const [showConfirm, setShowConfirm] = useState(false);
  const [selectedDate, setSelectedDate] = useState(
    new Date().toISOString().split('T')[0]
  );
  const [selectedTimeSlot, setSelectedTimeSlot] = useState<TimeSlot |
null>();
  const [availableDayss, setAvailableDayss] = useState<undefined | number[]>(
    undefined
  );

  console.log(selectedTimeSlot);
  console.log(match.params.id);

  useEffect(() => {
    setTimeout(() => {
      setAvailableDayss(availableDays);
    }, 2000);
  }, []);

  useEffect(() => {
```

```
    const fetchCoworkingInfo = async () => {
      const coworkingData = await getCoworkingInfoById(match.params.id);
      setCoworking(coworkingData);
    };

    fetchCoworkingInfo();

    const fetchWorkingPlaces = async () => {
      const workingPlacesData = await getWorkingPlaces(match.params.id);
      if (workingPlacesData.length) {
        const filteredWPAvailableDates = workingPlacesData.filter(
          (workingPlace) =>
            Boolean(Object.keys(workingPlace.availableDates || {}).length)
        );
        setWorkingPlaces(filteredWPAvailableDates);
        setSelectedDate(
          Object.keys(filteredWPAvailableDates[0].availableDates)[0]
        );

console.log(Object.keys(filteredWPAvailableDates[0].availableDates)[0]);
      } else {
        setWorkingPlaces(mockWorkingPlaces);
        setSelectedDate(Object.keys(mockWorkingPlaces[0].availableDates)[0]);
      }
    };

    fetchWorkingPlaces();
  }, [match.params.id]);

  const handleBookClick = (place: string) => {
    setBookingPlaceId(place);
    setShowConfirm(true);
  };

  const handleConfirm = () => {
    setShowConfirm(false);
    console.log(
      user.id,
      bookingPlaceId,
      coworking!.id,
      selectedDate,
      selectedTimeSlot
    );
```

```
  if (!bookingPlaceId!.startsWith('wp')) {
    createBooking(
      {
        userId: user.id,
        workingPlaceId: bookingPlaceId!,
        coworkingId: coworking!.id,
        timeSlot: selectedTimeSlot!,
      },
      [selectedDate]
    );
  }
  setWorkingPlaces((prevWPs) =>
    prevWPs.map((wp) => {
      if (wp.id !== bookingPlaceId) {
        return wp;
      } else {
        return {
          ...wp,
          availableDates: {
            ...wp.availableDates,
            [selectedDate]: wp.availableDates[selectedDate].filter(
              ({ startTime, endTime }) =>
                startTime.hour !== selectedTimeSlot!.startTime.hour &&
                endTime.hour !== selectedTimeSlot!.endTime.hour
            ),
          },
        };
      }
    })
  );
  setSelectedTimeSlot(null);
};

const handleDateChange = (e: any) => {
  e.preventDefault();
  setSelectedDate(e.detail.value);
  console.log(e.detail.value);
  e.detail.value !== selectedDate && setSelectedTimeSlot(null);
};

const handleTimeSlotChange = (e: any) => {
  console.log(e.detail.value);
  setSelectedTimeSlot(e.detail.value);
```

```
  };

  const availableDatesAndTimeSlots = workingPlaces.length
    ? (workingPlaces as WorkingPlace[]).reduce((acc, place) => {
        if (place.availableDates) {
          const datesAndTimeSlots = Object.entries(place.availableDates).map(
            ([date, timeSlots]) => ({
              date,
              timeSlots,
            })
          );
          return [...acc, ...datesAndTimeSlots];
        }
        return acc;
      }, [] as { date: string; timeSlots: TimeSlot[] }[])
    : mockedAvailableDatesAndTimeSlots;
  console.log(availableDatesAndTimeSlots);
  // const availableDatesAndTimeSlots = workingPlaces.length ?
workingPlaces.reduce((acc, place) => {
  //   if (place.availableDates) {
  //     return [...acc, ...place.availableDates];
  //   }
  //   return acc;
  // }, []) : [{ date: '2023-06-01', timeSlots: [{ startTime: { hour: '15',
minutes: '00' }, endTime: { hour: '17', minutes: '15' } }, { startTime: {
hour: '17', minutes: '30' }, endTime: { hour: '18', minutes: '45' } }] }];

  const availableYears = Array.from(
    new Set(
      availableDatesAndTimeSlots.map((item) =>
        new Date(item.date).getFullYear()
      )
    )
  );
  console.log(availableYears);

  const availableMonths = Array.from(
    new Set(
      availableDatesAndTimeSlots
        .filter(
          (item) =>
            new Date(item.date).getFullYear() ===
            new Date(selectedDate).getFullYear()
```

```
        )
        .map((item) => new Date(item.date).getMonth() + 1)
    )
  );
  console.log(availableMonths);

  const availableDays = Array.from(
    new Set(
      availableDatesAndTimeSlots
        .filter(
          (item) =>
            new Date(item.date).getFullYear() ===
              new Date(selectedDate).getFullYear() &&
            new Date(item.date).getMonth() === new
Date(selectedDate).getMonth()
        )
        .map((item) => new Date(item.date).getDate())
    )
  );
  // .sort()
  // .join(',');
  console.log(availableDays);

  const availableTimeSlots = selectedDate
    ? availableDatesAndTimeSlots.find((slot) => slot.date === selectedDate)
        ?.timeSlots || []
    : [];
  console.log(availableTimeSlots);

  const availablePlaces = selectedTimeSlot
    ? workingPlaces.length
      ? workingPlaces.filter(
          (place) =>
            Object.keys(place.availableDates).includes(selectedDate) &&
            place.availableDates[selectedDate].some(
              ({ startTime, endTime }) =>
                startTime.hour === selectedTimeSlot.startTime.hour &&
                endTime.hour === selectedTimeSlot.endTime.hour
            )
        )
      : mockWorkingPlaces
    : [];
```

```
const compareWith = (o1: TimeSlot, o2: TimeSlot) => {
  return o1 && o2
    ? o1.startTime.hour === o2.startTime.hour &&
        o1.endTime.hour === o2.endTime.hour
    : o1 === o2;
};

if (!coworking) return <IonLoading message={'Loading...'} />;

return (
  <IonPage>
    <IonHeader>
      <IonToolbar>
        <IonButtons slot='start'>
          <IonBackButton text='Previous' defaultHref='/Coworkings' />
        </IonButtons>
        <IonTitle>Book Place</IonTitle>
      </IonToolbar>
    </IonHeader>

    <IonContent>
      <IonGrid>
        <IonRow>
          <IonCol>
            <IonItem>
              <IonText>{coworking.name}</IonText>
            </IonItem>
          </IonCol>
        </IonRow>

        <IonRow>
          <IonCol>
            <IonItem>
              <IonText>{coworking.location}</IonText>
            </IonItem>
          </IonCol>
        </IonRow>

        <IonRow>
          <IonCol>
            <IonItem>
              <IonLabel position='floating'>Description</IonLabel>
              <IonTextarea
```

```
            value={coworking.description}
            readonly={true}
          ></IonTextarea>
        </IonItem>
      </IonCol>
    </IonRow>

    <IonRow>
      <IonCol>
        <div
          style={{
            width: '200px',
            height: '200px',
            border: '1px solid #000',
            borderRadius: '5px',
            overflow: 'hidden',
            margin: 'auto',
          }}
        >
          <Swiper modules={[Pagination]} pagination={true}>
            {coworking.imageUrls
              .filter((url) => url)
              .map((url, index) => (
                <SwiperSlide key={index}>
                  <IonImg
                    src={url}
                    alt={`Slide ${index}`}
                    style={{ width: '100%', height: '100%' }}
                  />
                </SwiperSlide>
              ))}
          </Swiper>
        </div>
      </IonCol>
    </IonRow>

    <IonItem lines='none'>
      <IonLabel>Date:</IonLabel>
      <IonDatetime
        presentation='date'
        value={selectedDate}
        onIonChange={handleDateChange}
        yearValues={availableYears}
```

```
                monthValues={availableMonths}
                dayValues={availableDays}
                min={new Date().toISOString().split('T')[0]}
              ></IonDatetime>
            </IonItem>

            {selectedDate && (
              <IonItem lines='none'>
                {/* <IonLabel>Time Slot:</IonLabel> */}
                <IonSelect
                  aria-label='time-slot'
                  placeholder='select time-slot'
                  compareWith={compareWith}
                  onIonChange={handleTimeSlotChange}
                  value={selectedTimeSlot}
                  style={{
                    margin: 'auto',
                    border: '1px solid #000',
                    borderRadius: '5px',
                    padding: '10px',
                    width: 'fit-content',
                  }}
                >
                  {availableTimeSlots.map(({ startTime, endTime }, index) => (
                    <IonSelectOption key={index} value={{ startTime, endTime
}}>
                      {startTime.hour}:{startTime.minute} - {endTime.hour}:
                      {endTime.minute}
                    </IonSelectOption>
                  ))}
                </IonSelect>
              </IonItem>
            )}

            {selectedTimeSlot && (
              <IonGrid>
                <IonRow>
                  {availablePlaces.map((place, index) => (
                    <IonCol size='6' key={index}>
                      <IonCard style={{ height: '150px', maxHeight: '150px' }}>
                        <IonCardHeader>
                          <IonCardSubtitle>{place.position}</IonCardSubtitle>
                        </IonCardHeader>
```

```
                <IonCardContent>
                  Seats: {place.seats} <br />
                  <small style={{ fontSize: '1.2' }}>
                    Price per hour: {place.pricePerHour}
                  </small>
                  <IonButton
                    size='small'
                    expand='full'
                    onClick={() => handleBookClick(place.id)}
                    style={{ marginTop: '20px' }}
                  >
                    Book
                  </IonButton>
                </IonCardContent>
              </IonCard>
            </IonCol>
          ))}
        </IonRow>
      </IonGrid>
    )}

    <IonAlert
      isOpen={showConfirm}
      trigger='book-alert'
      // header={'Confirm'}
      message={'Confirm your booking'}
      buttons={[
        {
          text: 'Cancel',
          role: 'cancel',
          cssClass: 'secondary',
          handler: () => {
            setShowConfirm(false);
          },
        },
        {
          text: 'Confirm',
          role: 'confirm',
          handler: handleConfirm,
        },
      ]}
    />
```

```
        </IonGrid>
      </IonContent>
    </IonPage>
  );
};

export default CoworkingPage;
```

Файл Coworkings.tsx

```
import React, { useEffect, useState } from 'react';
import { Virtuoso } from 'react-virtuoso';
import { RouteComponentProps, useParams } from 'react-router';
import {
  IonButtons,
  IonContent,
  IonHeader,
  IonMenuButton,
  IonPage,
  IonTitle,
  IonToolbar,
  IonLoading,
  useIonModal,
  IonIcon,
  IonInfiniteScroll,
  IonInfiniteScrollContent,
  InfiniteScrollCustomEvent,
  IonList,
} from '@ionic/react';
import { OverlayEventDetail } from '@ionic/core/components';
import { optionsOutline, analyticsOutline } from 'ionicons/icons';
import CoworkingItem from '../components/CoworkingItem';
import AlgoSettings from '../components/AlgoSettings';

import { CoworkingItemType, DocumentDataInterface, newSettings, RangeValue }
from '../types';
import { fetchCoworkings, updateUserSkills } from '../firebase/functions';
import { DocumentData } from 'firebase/firestore';

interface CoworkingsPageProps extends RouteComponentProps {
  user: DocumentData;
}
const Coworkings: React.FC<CoworkingsPageProps> = ({ user, history }) => {
```

```
const [data, setData] = useState<CoworkingItemType[]>([]);
const [userSkills, setUserSkills] = useState<string[]>(user.userSkills);
const [page, setPage] = useState(1);
const [priceRange, setPriceRange] = useState({ lower: 20, upper: 80 });
const [isLoading, setIsLoading] = useState(false);
const pageSize = 15;

const [present, dismiss] = useIonModal(AlgoSettings, {
  location: 'Kyiv, Pechersk',
  skills: userSkills,
  priceRange: priceRange,
  onDismiss: (
    data:
      | { location: string; priceRange: RangeValue; skills: string[] }
      | null
      | undefined,
    role: string
  ) => dismiss(data, role),
});

const openModal = () => {
  present({
    onWillDismiss: (ev: CustomEvent<OverlayEventDetail>) => {
      if (ev.detail.role === 'confirm') {
        setUserSkills(ev.detail.data.skills);
        updateUserSkills(user.id, ev.detail.data.skills);
        if (!ev.detail.data) { fetchCoworkingsBatch(ev.detail.data); }
      }
    },
  });
};


useEffect(() => {
  const unsubscribe = fetchCoworkings(setData);

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

const fetchCoworkingsBatch = async (data: any) => {
  setIsLoading(true);
```

```
    try {
      const res = await fetch(`api/algo-
restart/${location}/${userSkills[0]}/${priceRange.upper}/${page}`);
      const { coworkings } = await res.json();
      setData(prevData => [...prevData, ...coworkings]);
      setPage(page + 1);
    } catch (err) {
      console.error(err);
    }

    setIsLoading(false);
  };

  const handleEnd = async (ev: InfiniteScrollCustomEvent) => {
    if (data.length < 45) {
      await fetchCoworkingsBatch(userSkills);
    }

    ev.target.complete();
  };

  return (
    <IonPage>
      <IonHeader>
        <IonToolbar>
          <IonButtons slot='start'>
            <IonMenuButton />
          </IonButtons>
          <IonTitle>{ }</IonTitle>
          <IonIcon
            slot='end'
            icon={analyticsOutline}
            size='large'
            style={{ marginRight: '10px' }}
            onClick={() => openModal()}
          />
        </IonToolbar>
      </IonHeader>

      <IonContent fullscreen>
        <IonList>
          {data.map((coworking, index) => (
```

```
            <CoworkingItem
              key={coworking.id}
              coworking={coworking}
              networking={index === 1}
            // onButtonBookClick={handleButtonBookClick}
            />
          ))}
        </IonList>
        <IonInfiniteScroll threshold="5%" onIonInfinite={handleEnd}>
          <IonInfiniteScrollContent
            loadingSpinner="bubbles"
            loadingText="Loading more data..."
          >
          </IonInfiniteScrollContent>
        </IonInfiniteScroll>
      </IonContent>
    </IonPage>
  );
};

export default Coworkings;
```

Файл Login.tsx

```
import {
  IonContent,
  IonHeader,
  IonList,
  IonPage,
  IonText,
  IonTitle,
  IonToolbar,
} from '@ionic/react';
import React, { useState } from 'react';
import { IonGrid, IonRow, IonCol } from '@ionic/react';
import { personCircle } from 'ionicons/icons';
import { RouteComponentProps } from 'react-router-dom';
import {
  IonItem,
  IonLabel,
  IonInput,
  IonButton,
  IonIcon,
```

```
  IonAlert,
} from '@ionic/react';
import {
  auth,
  db,
  doc,
  getDoc,
  signInWithEmailAndPassword,
} from '../firebase/config';
import { getCoworkingId } from '../firebase/functions';
import { User } from '../types';

const validateEmail = (email: string) => {
  const re =
    // eslint-disable-next-line
    /^((?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-
]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-
\x09\x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-
9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-
9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-
9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-
\x09\x0b\x0c\x0e-\x7f])+)\]))$/;
  return re.test(String(email).toLowerCase());
};

const Login: React.FC<RouteComponentProps> = ({ history }) => {
  const [email, setEmail] = useState<string>('');
  const [password, setPassword] = useState<string>('');
  const [iserror, setIserror] = useState<boolean>(false);
  const [message, setMessage] = useState<string>('');

  const handleLogin = async () => {
    if (!email) {
      setMessage('Please enter a valid email');
      setIserror(true);
      return;
    }
    if (validateEmail(email) === false) {
      setMessage('Your email is invalid');
      setIserror(true);
      return;
    }
```

```
if (!password || password.length < 6) {
  setMessage('Please enter your password');
  setIserror(true);
  return;
}

try {
  const resp = await signInWithEmailAndPassword(auth, email, password);
  const uid = resp.user.uid;
  console.log(uid);

  const userRef = doc(db, 'users', uid);

  const firestoreDocument = await getDoc(userRef);
  if (firestoreDocument.exists()) {
    const user = firestoreDocument.data();
    if (!user.admin) {
      history.push('/Coworkings');
    } else {
      const coworkingId = await getCoworkingId(uid);
      console.log(coworkingId);

      if (coworkingId) {
        history.push('/WorkingPlaces', { coworkingId });
      } else {
        history.push('/addCoworking');
      }
    }
  } else {
    alert('User does not exist anymore.');
    return;
  }
} catch (error) {
  alert(error);
}

// const api = axios.create({
//   baseURL: `https://reqres.in/api`,
// });
// api
//   .post('/login', loginData)
//   .then((res) => {
//     history.push('/dashboard/' + email);
```

```
  //   })
  //   .catch((error) => {
  //     setMessage('Auth failure! Please create an account');
  //     setIserror(true);
  //   });
};

return (
  <IonPage>
    <IonHeader>
      <IonToolbar>
        <IonTitle>Login</IonTitle>
      </IonToolbar>
    </IonHeader>
    <IonContent fullscreen className='ion-padding ion-text-center'>
      <IonGrid>
        <IonRow>
          <IonCol>
            <IonAlert
              isOpen={iserror}
              onDidDismiss={() => setIserror(false)}
              cssClass='my-custom-class'
              header={'Error!'}
              message={message}
              buttons={['Dismiss']}
            />
          </IonCol>
        </IonRow>
        <IonRow>
          <IonCol>
            <IonIcon
              style={{ fontSize: '70px', color: '#0040ff' }}
              icon={personCircle}
            />
          </IonCol>
        </IonRow>
        <IonRow>
          <IonCol>
            <IonItem>
              <IonLabel position='floating'> Email</IonLabel>
              <IonInput
                type='email'
                value={email}
```

```
            autocomplete='email'
            onIonInput={(e) => setEmail(e.detail.value!)}
          ></IonInput>
        </IonItem>
      </IonCol>
    </IonRow>

    <IonRow>
      <IonCol>
        <IonItem>
          <IonLabel position='floating'> Password</IonLabel>
          <IonInput
            type='password'
            value={password}
            autocomplete='current-password'
            onIonInput={(e) => setPassword(e.detail.value!)}
          ></IonInput>
        </IonItem>
      </IonCol>
    </IonRow>
    <IonRow>
      <IonCol>
        {/* <p style={{ fontSize: 'small' }}>
          By clicking LOGIN you agree to our <a href='#'>Policy</a>
        </p> */}
        <IonButton expand='block' onClick={handleLogin}>
          Login
        </IonButton>
        <IonList>
          <IonItem
            routerLink={'/register'}
            routerDirection='none'
            lines='none'
            detail={false}
          >
            Don&apos;t have an account?  
            <IonText color='danger'> Sign up!</IonText>
          </IonItem>
        </IonList>
        {/* <p style={{ fontSize: 'medium' }}>
          Don't have an account? <a href='#'>Sign up!</a>
        </p> */}
      </IonCol>
```

```
          </IonRow>
        </IonGrid>
      </IonContent>
    </IonPage>
  );
};

export default Login;
```

Файл Register.tsx

```
import {
  IonCard,
  IonCardHeader,
  IonContent,
  IonHeader,
  IonList,
  IonPage,
  IonText,
  IonTitle,
  IonToggle,
  IonToolbar,
} from '@ionic/react';
import React, { useState } from 'react';

import { IonGrid, IonRow, IonCol } from '@ionic/react';
import {
  addCircleOutline,
  personCircle,
  removeCircleOutline,
} from 'ionicons/icons';
import { RouteComponentProps } from 'react-router-dom';
import {
  IonItem,
  IonLabel,
  IonInput,
  IonButton,
  IonIcon,
  IonAlert,
} from '@ionic/react';
import {
  auth,
```

```
  collection,
  createUserWithEmailAndPassword,
  db,
  doc,
  setDoc,
} from '../firebase/config';

function validateEmail(email: string) {
  const re =
    // eslint-disable-next-line
    /^((?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-
]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-
\x09\x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-
9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-
9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-
9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-
\x09\x0b\x0c\x0e-\x7f])+)\]))$/;
  return re.test(String(email).toLowerCase());
}
const Register: React.FC<RouteComponentProps> = ({ history }) => {
  const [isAdmin, setIsAdmin] = useState<boolean>(false);
  const [email, setEmail] = useState<string>('');
  const [userSkills, setUserSkills] = useState<string[]>([]);
  const [password, setPassword] = useState<string>('');
  const [iserror, setIserror] = useState<boolean>(false);
  const [message, setMessage] = useState<string>('');

  const handleRegister = async () => {
    if (!email) {
      setMessage('Please enter a valid email');
      setIserror(true);
      return;
    }
    if (validateEmail(email) === false) {
      setMessage('Your email is invalid');
      setIserror(true);
      return;
    }

    if (!password || password.length < 6) {
      setMessage('Please enter your password');
      setIserror(true);
      return;
```

```
    }

    // try {
    const resp = await createUserWithEmailAndPassword(auth, email, password);
    const uid = resp.user.uid;
    const data = isAdmin
      ? { id: uid, email, admin: true }
      : { id: uid, email, userSkills };
    const usersRef = collection(db, 'users');
    await setDoc(doc(usersRef, uid), data);
    if (!isAdmin) {
      history.push('/Coworkings');
    } else {
      history.push('/addCoworking');
    }
    // } catch (error) {
    //   alert(error);
    // }
  };

  const addSkill = () => {
    setUserSkills([...userSkills, '']);
  };

  const removeSkill = (index: number) => {
    const newSkills = [...userSkills];
    newSkills.splice(index, 1);
    setUserSkills(newSkills);
  };

  const handleSkillChange = (value: string, index: number) => {
    const newSkills = [...userSkills];
    newSkills[index] = value;
    setUserSkills(newSkills);
  };

  return (
    <IonPage>
      <IonHeader>
        <IonToolbar>
          <IonTitle>Register</IonTitle>
        </IonToolbar>
      </IonHeader>
```

```
<IonContent fullscreen className='ion-padding ion-text-center'>
  <IonGrid>
    <IonRow>
      <IonCol>
        <IonAlert
          isOpen={iserror}
          onDidDismiss={() => setIserror(false)}
          cssClass='my-custom-class'
          header={'Error!'}
          message={message}
          buttons={['Dismiss']}
        />
      </IonCol>
    </IonRow>
    <IonRow>
      <IonCol>
        <IonItem>
          <IonToggle
            onIonChange={(e) => setIsAdmin(!isAdmin)}
            justify='end'
          >
            coworking admin
          </IonToggle>
        </IonItem>
      </IonCol>
      <br />
      <br />
      <br />
      <br />
    </IonRow>
    <IonRow>
      <IonCol>
        <IonIcon
          style={{ fontSize: '70px', color: '#0040ff' }}
          icon={personCircle}
        />
      </IonCol>
    </IonRow>
    <IonRow>
      <IonCol>
        <IonItem>
          <IonLabel position='floating'> Email</IonLabel>
          <IonInput
```

```
            type='email'
            value={email}
            onIonChange={(e) => setEmail(e.detail.value!)}
          ></IonInput>
        </IonItem>
      </IonCol>
    </IonRow>
    <IonRow>
      <IonCol>
        <IonItem>
          <IonLabel position='floating'> Password</IonLabel>
          <IonInput
            type='password'
            value={password}
            onIonChange={(e) => setPassword(e.detail.value!)}
          ></IonInput>
        </IonItem>
      </IonCol>
    </IonRow>
    {!isAdmin &&
      userSkills &&
      userSkills.map((skill, index) => (
        <IonRow key={index}>
          <IonCol>
            <IonItem>
              <IonLabel>Skill {index + 1}</IonLabel>
              <IonInput
                value={skill}
                placeholder='Enter a skill'
                onIonChange={(e) =>
                  handleSkillChange(e.detail.value || '', index)
                }
              />
              <IonIcon
                slot='end'
                icon={removeCircleOutline}
                onClick={() => removeSkill(index)}
              />
            </IonItem>
          </IonCol>
        </IonRow>
      ))}
    {!isAdmin && (
```

```
            <div
              style={{
                display: 'flex',
                alignItems: 'center',
                justifyContent: 'center',
                paddingTop: '50px',
              }}
            >
              <IonLabel>Add another skill</IonLabel>
              <IonIcon
                icon={addCircleOutline}
                onClick={addSkill}
                size='large'
                style={{ marginLeft: '10px' }}
              />
            </div>
          )}
          <IonRow>
            <IonCol>
              {/* <p style={{ fontSize: 'small' }}>
                By clicking LOGIN you agree to our <a href='#'>Policy</a>
              </p> */}
              <IonButton expand='block' onClick={handleRegister}>
                Sign up
              </IonButton>
              <IonList>
                <IonItem
                  routerLink={'/login'}
                  routerDirection='none'
                  lines='none'
                  detail={false}
                >
                  Already registered?  
                  <IonText color='danger'> Sign in!</IonText>
                </IonItem>
              </IonList>
            </IonCol>
          </IonRow>
        </IonGrid>
      </IonContent>
    </IonPage>
  );
};
```

```
export default Register;
```

Файл WorkingPlaces.tsx

```
import React, { useState, useEffect, useRef, useCallback } from 'react';
import {
  IonContent,
  IonHeader,
  IonPage,
  IonTitle,
  IonToolbar,
  IonGrid,
  IonRow,
  IonCol,
  IonButton,
  IonIcon,
  useIonModal,
} from '@ionic/react';
import { OverlayEventDetail } from '@ionic/core/components';
import { add, timeOutline } from 'ionicons/icons';
import WorkingPlaceCard from '../components/WorkingPlaceCard'
import { WorkingPlace, AvailableDate, User, TimeSlot, CustomizedState } from
'../types';
import AvailabilityModal from '../components/AvailabilityModal';
import {
  getWorkingPlaces,
  createWorkingPlace,
  deleteWorkingPlace,
  addAvailableDateToWorkingPlaces,
  createBooking,
} from '../firebase/functions';
import { RouteComponentProps, useLocation } from 'react-router';
import { DocumentData } from 'firebase/firestore';

export const mockedAvailableDates = {
  '2023-06-01': [
    { startTime: { hour: '10', minute: '00' }, endTime: { hour: '12', minute:
'15' } },
    { startTime: { hour: '13', minute: '30' }, endTime: { hour: '15', minute:
'45' } },
```

```
    { startTime: { hour: '16', minute: '30' }, endTime: { hour: '18', minute:
'45' } },
  ],
  '2023-06-02': [
    { startTime: { hour: '09', minute: '00' }, endTime: { hour: '11', minute:
'15' } },
    { startTime: { hour: '12', minute: '30' }, endTime: { hour: '14', minute:
'45' } },
    { startTime: { hour: '15', minute: '30' }, endTime: { hour: '17', minute:
'45' } },
  ],
  '2023-06-03': [
    { startTime: { hour: '10', minute: '00' }, endTime: { hour: '12', minute:
'15' } },
    { startTime: { hour: '13', minute: '30' }, endTime: { hour: '15', minute:
'45' } },
    { startTime: { hour: '16', minute: '30' }, endTime: { hour: '18', minute:
'45' } },
  ],
  '2023-06-04': [
    { startTime: { hour: '11', minute: '00' }, endTime: { hour: '13', minute:
'15' } },
    { startTime: { hour: '14', minute: '30' }, endTime: { hour: '16', minute:
'45' } },
    { startTime: { hour: '17', minute: '30' }, endTime: { hour: '19', minute:
'45' } },
  ],
  '2023-06-05': [
    { startTime: { hour: '10', minute: '00' }, endTime: { hour: '12', minute:
'15' } },
    { startTime: { hour: '13', minute: '30' }, endTime: { hour: '15', minute:
'45' } },
    { startTime: { hour: '16', minute: '30' }, endTime: { hour: '18', minute:
'45' } },
  ],
  '2023-06-06': [
    { startTime: { hour: '09', minute: '00' }, endTime: { hour: '11', minute:
'15' } },
    { startTime: { hour: '12', minute: '30' }, endTime: { hour: '14', minute:
'45' } },
    { startTime: { hour: '15', minute: '30' }, endTime: { hour: '17', minute:
'45' } },
  ],
```

```
};

const getRandomSubobject = (obj: any, min: number, max: number) => {
  const keys = Object.keys(obj);
  const size = Math.floor(Math.random() * (max - min + 1)) + min;
  const randomKeys = Array.from({ length: size }, () => {
    const index = Math.floor(Math.random() * keys.length);
    return keys.splice(index, 1)[0];
  });
  return randomKeys.reduce((acc, key) => ({ ...acc, [key]: obj[key] }), {});
};

const positions = ['North wing', 'South wing', 'East wing', 'West wing',
'Center'];

export const mockWorkingPlaces = Array.from({ length:
Math.floor(Math.random() * 11) + 5 }, () => {
  const seats = Math.floor(Math.random() * 5) + 1;
  const pricePerHour = seats * 10 + Math.floor(Math.random() * 21) - 10;
  const randomAvailableDates = getRandomSubobject(mockedAvailableDates, 1,
5);
  return {
    id: `wp_${Math.random()}`,
    seats: seats,
    position: positions[Math.floor(Math.random() * positions.length)],
    pricePerHour: pricePerHour,
    availableDates: mockedAvailableDates
  };
});

interface WorkingPlacesPageProps extends RouteComponentProps {
  user: DocumentData;
}

const WorkingPlaces: React.FC<WorkingPlacesPageProps> = ({ user, location })
=> {
  const [coworkingId, setCoworkingId] = useState<string | null>(null);
  const [isEditing, setIsEditing] = useState<boolean | string>(false);
  const [selectedPlaces, setSelectedPlaces] = useState<string[]>([]);
  const [workingPlaces, setWorkingPlaces] = useState<WorkingPlace[]>([]);

  const state = location.state as CustomizedState;
```

```
useEffect(() => {
  setCoworkingId(state.coworkingId as string);
}, []);

const fetchWorkingPlaces = useCallback(async () => {
  if (coworkingId) {
    const data = await getWorkingPlaces(coworkingId);
    // const timestamp = serverTimestamp();

    if (data.length === 0) {
      setWorkingPlaces(mockWorkingPlaces);
    } else {
      setWorkingPlaces(data);
    }
  }
}, [coworkingId]);

useEffect(() => {
  if (coworkingId) {
    fetchWorkingPlaces();
  }
}, [coworkingId, fetchWorkingPlaces]);

const handleAddWorkingPlace = (e: any) => {
  e.preventDefault();
  if (!isEditing) {
    // const timestamp = serverTimestamp();
    const newWorkingPlace = {
      id: `wp_${Math.random()}`,
      seats: 1,
      position: 'North wing',
      pricePerHour: 10,
      // createdAt: timestamp,
      availableDates: {}
    };
    setWorkingPlaces((prevWorkingPlaces) => [
      ...prevWorkingPlaces,
      newWorkingPlace
    ]);
    console.log(newWorkingPlace);
    setIsEditing(newWorkingPlace.id);
    // window.scrollTo(0, document.body.scrollHeight);
  }
```

```
  };

  const handleSaveWorkingPlace = async (workingPlace: Omit<WorkingPlace,
'id'>) => {
    if (coworkingId) {
      await createWorkingPlace(workingPlace, coworkingId);
      fetchWorkingPlaces();

      if (workingPlaces.some(place => place.id.startsWith('wp_'))) {
        setSelectedPlaces(selectedPlaces.filter(id =>
!id.startsWith('wp_')));
      }
    }
  }

  const handleDeleteWorkingPlace = async (workingPlaceId: string) => {
    setWorkingPlaces(workingPlaces.filter((place) => place.id !==
workingPlaceId));
    setSelectedPlaces(selectedPlaces.filter(id => id !== workingPlaceId));
    if (isEditing) {
      setIsEditing(false);
    } else {
      await deleteWorkingPlace(workingPlaceId);
    }
    console.log(`Deleting working place: ${workingPlaceId}`);
  };

  const handlePlaceSelection = (placeId: string, isSelected: boolean) => {
    if (isSelected) {
      setSelectedPlaces([...selectedPlaces, placeId]);
    } else {
      setSelectedPlaces(selectedPlaces.filter((id) => id !== placeId));
    }
  };

  const [present, dismiss] = useIonModal(AvailabilityModal, {
    onDismiss: (
      data:
        | {
          dates: string[];
          startTime: string;
          endTime: string;
          availability: boolean;
```

```
        }
        | null
        | undefined,
      role: string
    ) => dismiss(data, role),
  });

  const openModal = () => {
    present({
      onWillDismiss: async (ev: CustomEvent<OverlayEventDetail>) => {
        if (ev.detail.role === 'confirm') {
          const { dates, startTime, endTime, availability }
            : { dates: string[], startTime: string, endTime: string,
availability: boolean } = ev.detail.data;
          const newStartTime = startTime.split(':');
          const newEndTime = endTime.split(':');
          const newTimeSlot: TimeSlot = {
            startTime: { hour: newStartTime[0], minute: newStartTime[1] },
            endTime: { hour: newEndTime[0], minute: newEndTime[1] },
          };
          console.log(ev.detail.data);
          console.log(selectedPlaces);
          if (availability) {
            await addAvailableDateToWorkingPlaces(selectedPlaces, { dates,
timeSlot: newTimeSlot });
          } else {
            const bookingPromises = selectedPlaces.map(place =>
createBooking({
              userId: user.id,
              coworkingId: coworkingId as string,
              timeSlot: newTimeSlot,
              workingPlaceId: place
            }, dates));

            await Promise.all(bookingPromises);
          }
          await fetchWorkingPlaces();
        }
      },
    });
  };

  return (
```

```
<IonPage>
  <IonHeader>
    <IonToolbar>
      <IonTitle>Working Places</IonTitle>
      <IonButton
        slot='end'
        size='small'
        fill='outline'
        disabled={selectedPlaces.length === 0}
        style={{ marginRight: '10px' }}
        onClick={() => openModal()}
      >
        <small>Set time</small>
        <IonIcon slot='end' icon={timeOutline} />
      </IonButton>
      <IonButton
        slot='end'
        size='small'
        fill='outline'
        disabled={Boolean(isEditing)}
        style={{ marginRight: '10px' }}
        onClick={handleAddWorkingPlace}
      >
        <small>Add place</small>
        <IonIcon slot='end' icon={add} />
      </IonButton>
    </IonToolbar>
  </IonHeader>
  <IonContent fullscreen>
    <IonGrid>
      <IonRow>
        {workingPlaces.map((place, index) => (
          <IonCol size='6' key={place.id}>
            <WorkingPlaceCard
              place={place}
              handleSaveWorkingPlace={handleSaveWorkingPlace}
              handleDeleteWorkingPlace={handleDeleteWorkingPlace}
              isEditing={
                Boolean(isEditing) &&
                (typeof isEditing === 'string'
                  ? isEditing === place.id
                  : true)
              }
```

```
                  checked={selectedPlaces.includes(place.id)}
                  setIsEditing={setIsEditing}
                  onSelectionChange={handlePlaceSelection}
                />
              </IonCol>
            ))}
          </IonRow>
        </IonGrid>
      </IonContent>
    </IonPage>
  );
};

export default WorkingPlaces;
```

Файл firebase/config.ts

```
import { initializeApp } from 'firebase/app';
import {
  getAuth,
  createUserWithEmailAndPassword,
  signInWithEmailAndPassword,
  onAuthStateChanged,
} from 'firebase/auth';
import {
  getFirestore,
  collection,
  doc,
  setDoc,
  getDoc,
  getDocs,
  query,
  orderBy,
  limit,
  where,
  onSnapshot,
  writeBatch,
  serverTimestamp,
  addDoc,
  deleteDoc,
  updateDoc,
  Timestamp,
```

```
  DocumentData,
} from '@firebase/firestore';

const firebaseConfig = {
  apiKey: 'AIzaSyAVSZ2Wyq_2il303-UBMCZvSjdQUsZgHyY',
  authDomain: 'diplom-b8df8.firebaseapp.com',
  projectId: 'diplom-b8df8',
  storageBucket: 'diplom-b8df8.appspot.com',
  messagingSenderId: '10000325506',
  appId: '1:10000325506:web:cb7b425a71d75014544d9b',
};

const app = initializeApp(firebaseConfig);
const auth = getAuth();

// Initialize Cloud Firestore and get a reference to the service
const db = getFirestore(app);

export {
  app,
  auth,
  db,
  doc,
  collection,
  setDoc,
  getDoc,
  getDocs,
  query,
  orderBy,
  limit,
  where,
  onSnapshot,
  addDoc,
  deleteDoc,
  updateDoc,
  writeBatch,
  createUserWithEmailAndPassword,
  signInWithEmailAndPassword,
  onAuthStateChanged,
  serverTimestamp,
  Timestamp,
  getAuth
};
```

Файл firebase/functions.ts

```
import {
  CoworkingItemType,
  WorkingPlace,
  AvailableDate,
  NewAvailableDatesTime,
  Booking,
  User,
  TimeSlot,
  CreateBookingData
} from '../types';
import {
  app,
  auth,
  db,
  doc,
  collection,
  setDoc,
  getDoc,
  getDocs,
  query,
  orderBy,
  limit,
  where,
  onSnapshot,
  addDoc,
  deleteDoc,
  updateDoc,
  createUserWithEmailAndPassword,
  signInWithEmailAndPassword,
  onAuthStateChanged,
  serverTimestamp,
  writeBatch,
} from './config';

export const createWorkingPlace = async (workingPlace: Omit<WorkingPlace,
'id' | 'createdAt'>, coworkingId: string) => {
  try {
    const timestamp = serverTimestamp();
    const workingPlaceRef = collection(db, 'workingPlaces');
```

```
    const wpRef = await addDoc(workingPlaceRef, {
      ...workingPlace,
      coworkingId,
      createdAt: timestamp,
      availableDates: {},
    });
    console.log('WorkingPlace created with ID: ', wpRef.id);
  } catch (e) {
    console.error('Error adding document: ', e);
  }
};

export const createBooking = async (bookingData: CreateBookingData, dates:
string[]) => {
  try {
    const bookingsRef = collection(db, 'bookings');
    const batch = writeBatch(db);

    const workingPlaceRef = doc(db, 'workingPlaces',
bookingData.workingPlaceId);
    const workingPlaceSnap = await getDoc(workingPlaceRef);

    if (!workingPlaceSnap.exists()) {
      throw new Error('Working place not found');
    }

    const workingPlaceData = workingPlaceSnap.data() as WorkingPlace;
    const availableDates: { [date: string]: TimeSlot[] } =
workingPlaceData.availableDates || {};

    for (const date of dates) {
      const bookingRef = doc(bookingsRef);
      if (availableDates[date]) {
        availableDates[date] = availableDates[date].filter(timeSlot =>
          !(timeSlot.startTime.hour === bookingData.timeSlot.startTime.hour
&&
            timeSlot.endTime.hour === bookingData.timeSlot.endTime.hour));

        if (availableDates[date].length === 0) {
          delete availableDates[date];
        }
      }
      batch.set(bookingRef, {
```

```
          ...bookingData,
          date,
          createdAt: serverTimestamp(),
          status: 'active',
        });
      }
      batch.update(workingPlaceRef, {
        availableDates: availableDates
      });

      await batch.commit();
      console.log('Bookings created for certain dates');

    } catch (error) {
      console.error('Error creating booking: ', error);
    }
  };


  export const deleteWorkingPlace = async (id: string) => {
    const workingPlaceRef = doc(db, 'workingPlaces', id);

    const bookingsQuery = query(
      collection(db, 'bookings'),
      where('workingPlaceId', '==', id),
      where('status', '==', 'active')
    );

    const bookingsSnapshot = await getDocs(bookingsQuery);

    const batch = writeBatch(db);


    if (bookingsSnapshot.docs?.length) {
      bookingsSnapshot.docs.forEach((bookingDoc) => {
        const bookingRef = doc(db, 'bookings', bookingDoc.id);
        batch.update(bookingRef, { status: 'canceled' });
      });
    }

    batch.delete(workingPlaceRef);

    return await batch.commit();
```

```
};

export const cancelBooking = async (bookingId: string) => {
  const bookingRef = doc(db, 'bookings', bookingId);
  const bookingSnapshot = await getDoc(bookingRef);

  if (!bookingSnapshot.exists()) {
    throw new Error('Booking does not exist!');
  }
  const booking = bookingSnapshot.data();
  const workingPlaceRef = doc(db, 'workingPlaces', booking.workingPlaceId);
  const workingPlaceSnapshot = await getDoc(workingPlaceRef);

  if (!workingPlaceSnapshot.exists()) {
    throw new Error('WP of booking is empty!');
  }
  const workingPlace = workingPlaceSnapshot.data();

  await updateDoc(bookingRef, { status: 'canceled' });

  const availableDates: { [date: string]: TimeSlot[] } =
workingPlace.availableDates || {};

  if (availableDates[booking.date]) {
    availableDates[booking.date].push(booking.timeSlot);
  } else {
    availableDates[booking.date] = [booking.timeSlot];
  }

  await updateDoc(workingPlaceRef, { availableDates });
};

export const getUserInfo = async (userId: string) => {
  const userRef = doc(db, 'users', userId);

  const userSnapshot = await getDoc(userRef);

  if (!userSnapshot.exists()) {
    throw new Error('User does not exist!');
  }

  const user = userSnapshot.data();
  if (!user.userSkills) return {
```

```
    email: user.email,
    userSkills: ['admin']
  };

  return {
    email: user.email,
    userSkills: user.userSkills
  };
};

export const updateUserSkills = async (userId: string, newSkills: string[])
=> {
  const userRef = doc(db, 'users', userId);

  await updateDoc(userRef, { userSkills: newSkills });
}

export const addAvailableDateToWorkingPlaces = async (workingPlaceIds:
string[], newAvailableDatesTime: NewAvailableDatesTime) => {
  const promises = workingPlaceIds.map(async (id) => {
    const workingPlaceRef = doc(db, 'workingPlaces', id);
    const workingPlaceSnapshot = await getDoc(workingPlaceRef);

    if (!workingPlaceSnapshot.exists()) {
      console.log('Working place does not exist with id: ' + id);
      return;
    }

    const workingPlace = workingPlaceSnapshot.data();
    const availableDates: { [date: string]: TimeSlot[] } =
workingPlace.availableDates || {};

    for (const date of newAvailableDatesTime.dates) {
      // const existingDateIndex = availableDates.findIndex(ad => ad.date ===
date);

      if (availableDates[date]) {
        // for (const existingTimeSlot of availableDates[date]) {
        //   const updatedTimeSlots = compareTimeSlots(existingTimeSlot,
newAvailableDatesTime.timeSlot);
        // }
        availableDates[date].push(newAvailableDatesTime.timeSlot);
      } else {
```

```
        availableDates[date] = [newAvailableDatesTime.timeSlot];
      }
    }

    await updateDoc(workingPlaceRef, { availableDates });
  });

  await Promise.all(promises);
};

export const getWorkingPlaces = async (coworkingId: string):
Promise<WorkingPlace[]> => {
  const workingPlaces = collection(db, 'workingPlaces');
  const q = query(workingPlaces, where('coworkingId', '==', coworkingId),
orderBy('createdAt'));
  const querySnapshot = await getDocs(q);
  if (querySnapshot.empty) {
    return [];
  }
  return querySnapshot.docs.map((doc) => ({ ...(doc.data() as WorkingPlace),
id: doc.id }));
}

export const getBookingsByWorkingPlace = async (workingPlaceId: string):
Promise<Booking[]> => {
  const bookings = collection(db, 'bookings');
  const q = query(bookings, where('workingPlaceId', '==', workingPlaceId),
orderBy('createdAt', 'desc'));
  const querySnapshot = await getDocs(q);
  if (querySnapshot.empty) {
    return [];
  }
  return querySnapshot.docs.map((doc) => ({ ...(doc.data() as Booking), id:
doc.id }));
}

export const getBookingsByUserId = async (userId: string): Promise<Booking[]>
=> {
  const bookings = collection(db, 'bookings');
  const q = query(bookings, where('userId', '==', userId),
orderBy('createdAt', 'desc'));
  const querySnapshot = await getDocs(q);
  if (querySnapshot.empty) {
```

```
    return [];
  }
  return querySnapshot.docs.map((doc) => ({ ...(doc.data() as Booking), id:
doc.id }));
}

export const getCoworkingId = async (userId: string) => {
  const q = query(collection(db, 'coworkings'), where('authorID', '==',
userId));
  const querySnapshot = await getDocs(q);
  if (!querySnapshot.empty) {
    const coworkingDataId = querySnapshot.docs[0].id;
    return coworkingDataId;
  } else {
    return null;
  }
}

export const getCoworkingInfoById = async (coworkingId: string) => {

  const coworkingRef = doc(db, 'coworkings', coworkingId);

  const coworkingSnapshot = await getDoc(coworkingRef);

  if (!coworkingSnapshot.exists()) {
    throw new Error('coworking does not exist!');
  }

  const coworking = coworkingSnapshot.data();

  return {
    id: coworkingId,
    name: coworking.name,
    location: coworking.location,
    description: coworking.description,
    imageUrls: coworking.imageUrls,
  };
}

export const fetchCoworkings = (setData:
React.Dispatch<React.SetStateAction<CoworkingItemType[]>>) => {
  const coworkingsRef = collection(db, 'coworkings');
  const q = query(coworkingsRef, orderBy('createdAt', 'desc'));
```

```
  const unsubscribe = onSnapshot(
    q,
    (querySnapshot) => {
      const newCoworkings: any[] = [];
      querySnapshot.forEach((doc) => {
        const coworking = doc.data();
        coworking.id = doc.id;
        newCoworkings.push(coworking);
      });
      setData(newCoworkings);
    },
    (error) => {
      console.log(error);
    }
  );

  return unsubscribe;
}
```

Файл Menu.tsx

```
import React from 'react';
import {
  IonContent,
  IonIcon,
  IonItem,
  IonLabel,
  IonList,
  IonListHeader,
  IonMenu,
  IonMenuToggle,
  IonNote,
} from '@ionic/react';

import { useLocation } from 'react-router-dom';
import {
  businessOutline,
  businessSharp,
  calendarOutline,
  calendarSharp,
} from 'ionicons/icons';
```

```
import './Menu.css';

interface AppPage {
  url: string;
  iosIcon: string;
  mdIcon: string;
  title: string;
}

const appPages: AppPage[] = [
  {
    title: 'Coworkings',
    url: '/Coworkings',
    iosIcon: businessOutline,
    mdIcon: businessSharp,
  },
  {
    title: 'Bookings',
    url: '/Bookings',
    iosIcon: calendarOutline,
    mdIcon: calendarSharp,
  },
];

const Menu: React.FC<{ email?: string, disabled: boolean }> = ({ email,
disabled }) => {
  const location = useLocation();

  return (
    <IonMenu contentId='main' type='overlay' disabled={disabled}>
      <IonContent>
        <IonList id='inbox-list'>
          <IonListHeader>Coworking diplom</IonListHeader>
          <IonNote>{email}</IonNote>
          {appPages.map((appPage, index) => {
            return (
              <IonMenuToggle key={index} autoHide={false}>
                <IonItem
                  className={
                    location.pathname === appPage.url ? 'selected' : ''
                  }
                  routerLink={appPage.url}
                  routerDirection='none'
```

```
                  lines='none'
                  detail={false}
                >
                  <IonIcon
                    aria-hidden='true'
                    slot='start'
                    ios={appPage.iosIcon}
                    md={appPage.mdIcon}
                  />
                  <IonLabel>{appPage.title}</IonLabel>
                </IonItem>
              </IonMenuToggle>
            );
          })}
        </IonList>
      </IonContent>
    </IonMenu>
  );
};

export default Menu;
```

Файл WorkingPlaceCard.tsx

```
import React, { useEffect, useLayoutEffect, useState } from 'react';
import {
  IonCard,
  IonButton,
  IonIcon,
  IonItem,
  IonCheckbox,
  IonInput,
  IonAlert,
  IonList,
  IonRow,
  IonCol,
} from '@ionic/react';
import {
  checkmarkOutline,
  listOutline,
  personCircleOutline,
  removeCircleOutline,
```

```
  trashOutline,
  closeCircleOutline,
  sadOutline,
  checkmarkDoneCircleOutline,
} from 'ionicons/icons';
import './WorkingPlaceCard.css';
import {
  cancelBooking,
  getBookingsByWorkingPlace,
  getUserInfo,
} from '../firebase/functions';
import { Booking, User, WorkingPlace, WithRequired, TimeSlot } from
'../types';

export const bookingsMock: Booking[] = [
  {
    id: 'bo1',
    userId: 'u1',
    workingPlaceId: 'wp1',
    coworkingId: 'co1',
    status: 'active',
    date: '2023-06-01',
    timeSlot: {
      startTime: { hour: '09', minute: '00' },
      endTime: { hour: '17', minute: '00' },
    },
  },
  {
    id: 'bo2',
    userId: 'u2',
    workingPlaceId: 'wp2',
    coworkingId: 'co2',
    status: 'active',
    date: '2023-06-02',
    timeSlot: {
      startTime: { hour: '10', minute: '30' },
      endTime: { hour: '15', minute: '30' },
    },
  },
  {
    id: 'bo3',
    userId: 'u3',
    workingPlaceId: 'wp3',
```

```
      coworkingId: 'co3',
      status: 'completed',
      date: '2023-06-03',
      timeSlot: {
        startTime: { hour: '08', minute: '00' },
        endTime: { hour: '14', minute: '00' },
      },
    },
  ];

  const WorkingPlaceCard: React.FC<{
    place: WorkingPlace;
    handleSaveWorkingPlace: (workingPlace: Omit<WorkingPlace, 'id'>) => void;
    handleDeleteWorkingPlace: (workingPlaceId: string) => void;
    isEditing: boolean;
    checked: boolean;
    setIsEditing: React.Dispatch<React.SetStateAction<boolean | string>>;
    onSelectionChange: (placeId: string, isSelected: boolean) => void;
  }> = ({
    place,
    handleSaveWorkingPlace,
    handleDeleteWorkingPlace,
    isEditing = false,
    checked,
    setIsEditing,
    onSelectionChange,
  }) => {
    const [viewBookings, setViewBookings] = useState(false);
    const [isSelected, setIsSelected] = useState(checked);
    const [seats, setSeats] = useState(place.seats);
    const [position, setPosition] = useState(place.position);
    const [pricePerHour, setPricePerHour] = useState(place.pricePerHour);
    const [availableDates, setAvailableDates] = useState<{
      [date: string]: TimeSlot[];
    }>(place.availableDates);
    const [bookings, setBookings] = useState<Booking[]>(bookingsMock);
    const [showUserInfo, setShowUserInfo] = useState(false);
    const [showDeleteAlert, setShowDeleteAlert] = useState(false);

    const [bookingUser, setBookingUser] = useState<Omit<
      WithRequired<User, 'userSkills'>,
      'id'
    > | null>(null);
```

```
useEffect(() => {
  const fetchBookings = async () => {
    if (viewBookings) {
      const data = await getBookingsByWorkingPlace(place.id);
      if (data.length) {
        setBookings(data);
      } else {
        setBookings(bookingsMock);
      }
    }
  };

  fetchBookings();
}, [place.id, place.availableDates, viewBookings]);

const handleDelete = async () => {
  if (!isEditing) {
    setShowDeleteAlert(true);
  } else {
    handleDeleteWorkingPlace(place.id);
  }
};

const handleConfirmDelete = async () => {
  handleDeleteWorkingPlace(place.id);
  console.log(`Deleting working place: ${place.id}`);
  setShowDeleteAlert(false);
};

const handleCancelBooking = async (bookingId: string) => {
  console.log('Canceling booking: ', bookingId);
  if (!bookingId.startsWith('bo')) {
    await cancelBooking(bookingId);
    setAvailableDates((prevAvailableDates) => {
      const date = bookings.find((booking) => booking.id === bookingId)
        ?.date as string;
      const timeSlot = bookings.find((booking) => booking.id === bookingId)
        ?.timeSlot as TimeSlot;
      if (prevAvailableDates[date as string]) {
        prevAvailableDates[date].push(timeSlot);
      } else {
        prevAvailableDates[date] = [timeSlot];
```

```
        }
        return prevAvailableDates;
      });
    }
    setBookings((prevBookings) =>
      prevBookings.map((booking) =>
        booking.id === bookingId ? { ...booking, status: 'canceled' } :
booking
      )
    );
  };

  const handleConfirmSave = async () => {
    await handleSaveWorkingPlace({
      seats,
      position,
      pricePerHour,
      availableDates: {},
    });
    setIsEditing(false);
  };

  const handleCheckboxChange = (e: any) => {
    const isChecked = e.target.checked;
    setIsSelected(isChecked);
    onSelectionChange(place.id, isChecked);
  };

  const handleUserIconClick = async (userId: string) => {
    let userInfo = null;
    try {
      userInfo = await getUserInfo(userId);
    } catch (error) {
      console.log(`Failed to get user info: ${error}`);
      userInfo = {
        email: 'admin@gmail.com',
        userSkills: ['db', 'infra', 'ai/ml'],
      };
    }

    setBookingUser(userInfo);
    setShowUserInfo(true);
  };
```

```
return (
  <IonCard className={`working-place-card`}>
    <div className='button-bar'>
      {!isEditing && (
        <>
          <IonButton
            fill='clear'
            size='small'
            onClick={() => setViewBookings(!viewBookings)}
          >
            <IonIcon icon={listOutline} />
          </IonButton>
          <IonCheckbox
            onIonChange={handleCheckboxChange}
            checked={isSelected}
          />
        </>
      )}
      <IonButton
        className={isEditing ? 'button-delete' : ''}
        fill='clear'
        size='small'
        onClick={handleDelete}
      >
        <IonIcon icon={removeCircleOutline} />
      </IonButton>
    </div>
    {viewBookings ? (
      <div
        style={{
          display: 'flex',
          flexDirection: 'column',
          justifyContent: 'space-between',
        }}
      >
        {Boolean(Object.keys(availableDates).length) && (
          <IonRow
            style={{
              maxHeight: '70px',
              overflow: 'auto',
              border: '1px solid black',
              padding: '0px',
```

```
                marginBottom: '2px',
              }}
            >
              {Object.entries(availableDates).map(([date, timeSlots]) => (
                <IonCol size='6' key={date}>
                  <div className='ion-item-labell'>{date}</div>
                  {timeSlots.map((timeSlot, tsIndex) => (
                    <div
                      className='ion-item-content'
                      key={tsIndex}

>{`${timeSlot.startTime.hour}:${timeSlot.startTime.minute} -
${timeSlot.endTime.hour}:${timeSlot.endTime.minute}`}</div>
                  ))}
                </IonCol>
              ))}
            </IonRow>
          )}

          <IonList
            style={{
              maxHeight: '165px',
              overflow: 'auto',
              border: '1px solid black',
              padding: '0',
            }}
          >
            {bookings.map(
              ({
                id,
                userId,
                date,
                status,
                timeSlot: { startTime, endTime },
              }) => (
                <IonItem key={id} className='ion-item-booking ion-item-
label'>
                  <IonIcon
                    size='small'
                    color='secondary'
                    icon={personCircleOutline}
                    onClick={() => handleUserIconClick(userId)}
                  />
```

```
                  <small
                    style={{
                      marginLeft: '14px',
                      marginRight: '7px',
                      width: '20vw',
                    }}
                  >{`${date} ${startTime.hour}:${startTime.hour} -
${endTime.hour}:${endTime.minute}`}</small>
                  {status === 'active' ? (
                    <IonIcon
                      size='small'
                      color='danger'
                      icon={trashOutline}
                      onClick={() => handleCancelBooking(id)}
                    />
                  ) : status === 'canceled' ? (
                    <IonIcon size='small' color='medium' icon={sadOutline} />
                  ) : status === 'completed' ? (
                    <IonIcon
                      size='small'
                      color='success'
                      icon={checkmarkDoneCircleOutline}
                    />
                  ) : null}
                </IonItem>
              )
            )}
          </IonList>
          <IonAlert
            isOpen={showUserInfo}
            onDidDismiss={() => setShowUserInfo(false)}
            subHeader={bookingUser ? `${bookingUser.email}` : ''}
            message={
              bookingUser ? `Skills: ${bookingUser.userSkills.join('\n')}` :
''
            }
            buttons={['OK']}
          />
        </div>
      ) : (
        <>
          {isEditing ? (
            <>
```

```
                <IonItem className='ion-item-label'>
                  <IonInput
                    className='ion-item-contentt'
                    type='number'
                    label={'Seat(s):'}
                    labelPlacement='stacked'
                    placeholder='enter number'
                    value={seats}
                    onIonInput={(e) => setSeats(Number(e.detail.value!))}
                  />
                </IonItem>
                <IonItem className='ion-item-label'>
                  <IonInput
                    className='ion-item-contentt'
                    label={'Position:'}
                    labelPlacement='stacked'
                    placeholder='enter text'
                    value={position}
                    onIonInput={(e) => setPosition(e.detail.value!)}
                  />
                </IonItem>
                <IonItem className='ion-item-label'>
                  <IonInput
                    className='ion-item-contentt'
                    type='number'
                    value={pricePerHour}
                    label={'Price per hour:'}
                    labelPlacement='stacked'
                    placeholder='enter number'
                    onIonInput={(e) =>
  setPricePerHour(Number(e.detail.value!))}
                  />
                </IonItem>
                <IonButton
                  className='button-confirm'
                  size='small'
                  onClick={handleConfirmSave}
                >
                  <IonIcon icon={checkmarkOutline} />
                </IonButton>
              </>
            ) : (
              <>
```

```
              <IonItem className='ion-item-label'>
                <small>Seat(s): </small> 
                <span className='ion-item-content'>{place.seats}</span>
              </IonItem>
              <IonItem className='ion-item-label'>
                <small>Position:</small> 
                <span className='ion-item-content'>{place.position}</span>
              </IonItem>
              <IonItem className='ion-item-label'>
                <small>Price per hour:</small> 
                <span className='ion-item-
content'>{place.pricePerHour}$</span>
              </IonItem>
            </>
          )}
        </>
      )}
      <IonAlert
        isOpen={showDeleteAlert}
        onDidDismiss={() => setShowDeleteAlert(false)}
        header={'Confirm Delete'}
        message={'delete this working place?'}
        className='custom-alert'
        buttons={[
          {
            text: 'Cancel',
            role: 'cancel',
            cssClass: 'secondary',
            handler: () => setShowDeleteAlert(false),
          },
          {
            text: 'Delete',
            role: 'destructive',
            handler: handleConfirmDelete,
          },
        ]}
      />
    </IonCard>
  );
};

export default WorkingPlaceCard;
```

Файл WorkingPlaceCard.css

```
.working-place-card {
  height: 270px;
  margin: 0px;
  padding: 3px;
}

.button-bar {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 0px;
}

.button-delete {
  margin-left: auto
}

.ion-item-label {
  color: #1a1a1a;
  font-size: 1em;
}

.ion-item-booking {
  display: flex;
  flex-direction: column-reverse;
  /* justify-content: space-between; */
  align-items: center;
}

.ion-item-content {
  color: #022164;
  font-size: 0.75em;
  text-align: center;
}

.ion-item-contentt {
  color: #050f25;
  font-size: 0.85em;
  /* text-align: center; */
}
```

```
.ion-item-labell {
  text-align: center;
  color: rgb(241, 83, 21);
  font-size: 0.85em;
}

.button-confirm {
  margin: auto;
  margin-top: 20px;
  display: block;
  width: 50%;
  border-radius: 5px;
}

ion-alert.custom-alert {
  --backdrop-opacity: 0.7;
}

.custom-alert .alert-button-group {
  padding: 8px;
  justify-content: space-between;
}

button.alert-button.alert-button-confirm {
  background-color: var(--ion-color-success);
  color: var(--ion-color-success-contrast);
}

.md button.alert-button.alert-button-confirm {
  border-radius: 4px;
}

.ios .custom-alert button.alert-button {
  border: 0.55px solid rgba(var(--ion-text-color-rgb, 0, 0, 0), 0.2);
}

.ios button.alert-button.alert-button-cancel {
  border-right: 0;
  border-bottom-left-radius: 13px;
  border-top-left-radius: 13px;
}
```

```
.ios button.alert-button.alert-button-confirm {
  border-bottom-right-radius: 13px;
  border-top-right-radius: 13px;
}
```

Файл CoworkingItem.tsx

```
import React from 'react';
import {
  IonCard,
  IonCardHeader,
  IonCardTitle,
  IonCardContent,
  IonBadge,
  IonImg,
  IonLabel,
  IonButton,
  IonRow,
  IonCol,
  IonItem,
  IonCardSubtitle,
  IonIcon,
} from '@ionic/react';
import { Swiper, SwiperSlide } from 'swiper/react';
import { Pagination } from 'swiper';
import { CoworkingItemType } from '../types';

import 'swiper/css';
import 'swiper/css/pagination';
import './CoworkingItem.css'; // добавлен файл со стилями

import { arrowForwardCircleOutline } from 'ionicons/icons';
interface CoworkingItemProps {
  coworking: CoworkingItemType;
  networking: boolean;
  // onButtonBookClick: (coworking: CoworkingItemType) => void;
}

const CoworkingItem: React.FC<CoworkingItemProps> = ({
  coworking: { id, name, location, description, imageUrls },
  networking,
  // onButtonBookClick
```

```
}) => {
  return (
    <IonCard className='coworking-card'>
      <IonCardHeader>
        <IonRow>
          <IonCol size='6'>
            <Swiper
              modules={[Pagination]}
              pagination={true}
              className='image-swiper'
            >
              {imageUrls.map((imageUrl, index) => (
                <SwiperSlide key={index}>
                  <IonImg className='coworking-image' src={imageUrl} />
                </SwiperSlide>
              ))}
            </Swiper>
          </IonCol>
          <IonCol size='6' className='coworking-info'>
            <IonCardTitle className='coworking-title'>{name}</IonCardTitle>
            <IonCardSubtitle>{location}</IonCardSubtitle>
            {networking && (
              <IonBadge className='badge-networking' color='primary'>
                <small>Networking potential</small>
              </IonBadge>
            )}
            <IonButton
              // onClick={e => onButtonBookClick(
              //   {
              //     id,
              //     name,
              //     location,
              //     imageUrls,
              //     description,
              //     networking
              //   })}
              routerLink={`/Coworkings/${id}`}
              className='book-button'
              size='small'
            >
              <IonLabel>Book</IonLabel>  
              <IonIcon icon={arrowForwardCircleOutline} size='small' />
            </IonButton>
```

```
          </IonCol>
        </IonRow>
      </IonCardHeader>
    </IonCard>
  );
};

export default CoworkingItem;
```

Файл CoworkingItem.css

```
.coworking-title {
  font-size: 1.2rem;
}

.coworking-card {
  height: 150px;
}

.coworking-image {}

.image-swiper {
  width: 120px;
  height: 120px;
}

.coworking-info {
  display: flex;
  flex-direction: column;
  justify-content: space-between;
}

.badge-networking {
  background-color: limegreen;
  width: 120px;
  margin-top: 5px;
}

.book-button {
  align-self: baseline;
}
```

Файл AvailabilityModal.tsx

```
import React, { useState, useRef } from 'react';
import {
  IonButton,
  IonModal,
  IonContent,
  IonInput,
  IonIcon,
  IonRange,
  IonLabel,
  IonPage,
  IonHeader,
  IonToolbar,
  IonButtons,
  IonTitle,
  IonItem,
  IonCard,
  IonCardHeader,
  IonDatetime,
  IonRadioGroup,
  IonRadio,
  IonCol,
  IonRow,
  IonToast,
} from '@ionic/react';
import { addCircleOutline, alertCircleOutline, removeCircleOutline } from
'ionicons/icons';
import './AvailabilityModal.css'

interface BookingModalProps {
  onDismiss: (
    data?:
      | {
        dates: string[];
        startTime: string;
        endTime: string;
        availability: boolean;
      }
      | null
      | undefined,
    role?: string
```

```
  ) => void;
}

const AvailabilityModal: React.FC<BookingModalProps> = ({ onDismiss }) => {
  const [pickedDates, setPickedDates] = useState<string[]>([]);
  const [userStartTime, setStartTime] = useState('10:00');
  const [userEndTime, setEndTime] = useState('18:00');
  const [availability, setAvailability] = useState<boolean>(true);
  const [showToast, setShowToast] = useState(false);
  const datetimeRef = useRef<HTMLIonDatetimeElement>(null);

  const handleDateChange = (e: any) => {
    setPickedDates(e.detail.value);
    console.log(e.detail.value);
  };

  const handleStartTimeChange = (e: any) => {
    console.log(e.detail.value);

    setStartTime(e.detail.value);
  };

  const handleEndTimeChange = (e: any) => {
    setEndTime(e.detail.value);
  };

  return (
    <IonPage>
      <IonHeader>
        <IonToolbar>
          <IonButtons slot='start'>
            <IonButton color='medium' onClick={() => onDismiss(null,
'cancel')}>
              Cancel
            </IonButton>
          </IonButtons>
          <IonTitle>Set working places time</IonTitle>
          <IonButtons slot='end'>
            <IonButton
              onClick={() => {
                if (pickedDates.length === 0) {
                  setShowToast(true);
                } else {
```

```
                onDismiss(
                  {
                    dates: pickedDates,
                    startTime: userStartTime,
                    endTime: userEndTime,
                    availability: availability,
                  },
                  'confirm'
                );
              }
            }}
            strong={true}
          >
            Confirm
          </IonButton>
        </IonButtons>
      </IonToolbar>
    </IonHeader>
    <IonContent>
      <IonItem>
        <IonDatetime
          ref={datetimeRef}
          presentation='date'
          multiple={true}
          min={new Date().toISOString().split('T')[0]}
          onIonChange={handleDateChange}
        ></IonDatetime>
      </IonItem>
      <IonRow>
        <IonCol size="6">
          <div className='container'>
            <label>Start Time</label>
            <div>
              <IonDatetime
                value={userStartTime}
                presentation='time'
                hour-cycle='h23'
                onIonChange={handleStartTimeChange}
              ></IonDatetime>
            </div>
          </div>
        </IonCol>
        <IonCol size="6">
```

```
            <div className='container'>
              <label>End Time</label>
              <div>
                <IonDatetime
                  value={userEndTime}
                  presentation='time'
                  hour-cycle='h23'
                  onIonChange={handleEndTimeChange}
                ></IonDatetime>
              </div>
            </div>
          </IonCol>
        </IonRow>
        <IonItem>
          <IonLabel>Choose type:</IonLabel>
          <IonRadioGroup
            value={availability}
            onIonChange={(e) => setAvailability(e.detail.value)}
          >
            <IonItem>
              <IonLabel>Available</IonLabel>
              <IonRadio slot='start' value={true} />
            </IonItem>
            <IonItem>
              <IonLabel>Booked</IonLabel>
              <IonRadio slot='start' value={false} />
            </IonItem>
          </IonRadioGroup>
        </IonItem>
        <IonToast
          isOpen={showToast}
          onDidDismiss={() => setShowToast(false)}
          message="Choose date(s) before confirming"
          duration={3000}
          buttons={[
            {
              text: 'OK',
              role: 'cancel',
            },
          ]}
          cssClass="custom-toast"
          icon={alertCircleOutline}
        />
```

```
      </IonContent>
    </IonPage>
  );
};

export default AvailabilityModal;
```

Файл AlgoSettings.tsx

```
import React, { useState, useEffect } from 'react';
import {
  IonButton,
  IonModal,
  IonContent,
  IonInput,
  IonIcon,
  IonRange,
  IonLabel,
  IonPage,
  IonHeader,
  IonToolbar,
  IonButtons,
  IonTitle,
  IonItem,
  IonCard,
  IonCardHeader,
} from '@ionic/react';
import { addCircleOutline, removeCircleOutline } from 'ionicons/icons';
import { RangeValue, newSettings } from '../types';
import './AlgoSettings.css';

interface AlgoSettingsProps {
  location: string;
  skills: string[];
  priceRange: RangeValue;
  // onAlgorithmRestart: (newSettings: newSettings) => void;
  onDismiss: (
    data?:
      | { location: string; priceRange: RangeValue; skills: string[] }
      | null
      | undefined,
    role?: string
```

```
  ) => void;
}

const AlgoSettings: React.FC<AlgoSettingsProps> = ({
  location,
  skills,
  priceRange,
  // onAlgorithmRestart,
  onDismiss,
}) => {
  const [userLocation, setUserLocation] = useState(location);
  const [userSkills, setUserSkills] = useState(skills);
  const [userPriceRange, setUserPriceRange] =
useState<RangeValue>(priceRange);

  useEffect(() => {
    setUserLocation(location);
    setUserSkills(skills);
    setUserPriceRange(priceRange);
  }, [location, skills, priceRange]);

  const addSkill = () => {
    setUserSkills([...userSkills, '']);
  };

  const removeSkill = (index: number) => {
    const newSkills = [...userSkills];
    newSkills.splice(index, 1);
    setUserSkills(newSkills);
  };

  const handleSkillChange = (value: string, index: number) => {
    const newSkills = [...userSkills];
    newSkills[index] = value;
    setUserSkills(newSkills);
  };

  return (
    <IonPage>
      <IonHeader>
        <IonToolbar>
          <IonButtons slot='start'>
```

```
            <IonButton color='medium' onClick={() => onDismiss(null,
'cancel')}>
              Cancel
            </IonButton>
          </IonButtons>
          <IonTitle>Algo Settings</IonTitle>
          <IonButtons slot='end'>
            <IonButton
              onClick={() =>
                onDismiss(
                  {
                    location: userLocation,
                    priceRange: userPriceRange,
                    skills: userSkills,
                  },
                  'confirm'
                )
              }
              strong={true}
            >
              Confirm
            </IonButton>
          </IonButtons>
        </IonToolbar>
      </IonHeader>
      <IonContent>
        <IonCard>
          <IonCardHeader>
            <IonLabel>Location</IonLabel>
          </IonCardHeader>
          <IonItem lines='none'>
            <IonInput
              value={userLocation}
              placeholder='Enter your location'
              // className='input-location'
              onIonChange={(e) => setUserLocation(e.detail.value || '')}
            />
          </IonItem>
        </IonCard>
        <IonCard>
          <IonCardHeader>
            <IonLabel>Price Range</IonLabel>
          </IonCardHeader>
```

```
          <IonItem lines='none'>
            <IonRange
              aria-label='Dual Knobs Range'
              dualKnobs={true}
              value={{
                lower: userPriceRange?.lower ?? 20,
                upper: userPriceRange?.upper ?? 80,
              }}
              onIonInput={(e) =>
                setUserPriceRange(e.detail.value as RangeValue)
              }
            >
              <IonLabel slot='start'>{userPriceRange?.lower ?? 0}</IonLabel>
              <IonLabel slot='end'>{userPriceRange?.upper ?? 100}</IonLabel>
            </IonRange>
          </IonItem>
        </IonCard>
        {userSkills &&
          userSkills.map((skill, index) => (
            <IonCard key={index}>
              <IonCardHeader>
                <IonLabel>Skill {index + 1}</IonLabel>
              </IonCardHeader>
              <IonItem lines='none'>
                <IonInput
                  value={skill}
                  placeholder='Enter a skill'
                  onIonInput={(e) =>
                    handleSkillChange(e.detail.value || '', index)
                  }
                />
                <IonIcon
                  slot='end'
                  // className='icon-end'
                  icon={removeCircleOutline}
                  onClick={() => removeSkill(index)}
                />
              </IonItem>
            </IonCard>
          ))}
        <div className='add-skill'>
          <IonLabel>Add another skill</IonLabel>
          <IonIcon
```

```
            icon={addCircleOutline}
            onClick={addSkill}
            size='large'
            style={{ marginLeft: '10px' }}
          />
        </div>{' '}
      </IonContent>
    </IonPage>
  );
};

export default AlgoSettings;
```

Файл functions/algo.js

```
import { onRequest } from 'firebase-functions/v2/https';
import { initializeApp } from 'firebase-admin/app';
import { getFirestore } from 'firebase-admin/firestore';
import * as cron from 'node-cron';

initializeApp();

cron.schedule('0 0 */2 * * *', async () => {
  const bookingsSnap = await getFirestore()
    .collection('bookings')
    .where('status', '==', 'active')
    .get();

  const promises = [];

  bookingsSnap.forEach((doc) => {
    const data = doc.data();
    const bookingDate = new Date(data.date);
    const bookingEndTime = new Date(
      bookingDate.getFullYear(),
      bookingDate.getMonth(),
      bookingDate.getDate(),
      data.timeSlot.endTime.hour,
      data.timeSlot.endTime.minute
    );

    if (bookingEndTime < new Date()) {
```

```
      promises.push(
        getFirestore()
          .collection('bookings')
          .doc(doc.id)
          .update({ status: 'completed' })
      );
    }
  });

  await Promise.all(promises);
});

export const updateBookings = onRequest(async (req, res) => {
  try {
    res.send('Cron job scheduled to update booking status.');
  } catch (error) {
    console.error(error);
    res.status(500).send(error);
  }
});

export const getCoworkings = onRequest(async (req, res) => {
  const userId = req.query.userId;
  const page = req.query.page || null;
  const location = req.query.location || null;
  const newSkills = req.query.newSkills?.split(',') || null;

  const db = getFirestore();

  // Fetch the user's data from Firestore
  const userRef = db.collection('users').doc(userId);
  const userSnapshot = await userRef.get();
  let userData = userSnapshot.data();

  if (!userData) {
    userData = {};
  }

  if (!userData.location) {
    await userRef.set({ ...userData, location: 'Kyiv, Khreshchatyk 1' });
    userData.location = 'Kyiv, Khreshchatyk 1';
  }
```

```
  if (location) {
    await userRef.update({ location });
    userData.location = location;
  }

  if (newSkills) {
    await userRef.update({ skills: newSkills });
    userData.skills = newSkills;
  }

  let coworkingsSnap = await db.collection('coworkings').get();

  let coworkings = coworkingsSnap.docs.map((doc) => doc.data());

  if (page) {
    const half = Math.ceil(coworkings.length / 2);
    if (page === '1') {
      coworkings = coworkings.slice(0, half);
    } else if (page === '2') {
      coworkings = coworkings.slice(half, coworkings.length);
    }
  }

  res.json({ coworkings });
});

const natural = require('natural');
const { WordNet, WordTokenizer } = natural;
const wordnet = new WordNet();
const tokenizer = new WordTokenizer();
const util = require('util');

const wordnetLookup = util.promisify(wordnet.lookup.bind(wordnet));

async function lemmatize(word) {
  const results = await wordnetLookup(word);

  if (results.length > 0) {
    return results[0].lemma;
  } else {
    return word;
  }
}
```

```
async function lemmatizeSkills(skills) {
  const lemmatizedSkills = [];

  for (const skill of skills) {
    const tokens = tokenizer.tokenize(skill);
    const lemmatizedTokens = await Promise.all(
      tokens.map((token) => lemmatize(token))
    );
    lemmatizedSkills.push(lemmatizedTokens.join(' '));
  }

  return lemmatizedSkills;
}

const calculateMatchScore = (userSkills, coworkingVisitors) => {
  let totalScore = 0;

  for (const visitor of coworkingVisitors) {
    let sharedSkills = 0;

    for (const skill of userSkills) {
      if (visitor.skills.includes(skill)) {
        sharedSkills++;
      }
    }
    totalScore += sharedSkills * visitor.visitFrequency;
  }
  return totalScore;
};

async function sortCoworkingsByNetworkingPotential(userData, coworkingData) {
  const lemmatizedUserSkills = await lemmatizeSkills(userData.skills);

  return coworkingData
    .map((coworking) => {
      const matchScore = calculateMatchScore(
        lemmatizedUserSkills,
        coworking.visitors
      );
      const eventsScore = coworking.events.filter((event) =>
        lemmatizedUserSkills.includes(event.topic)
      ).length;
```

```
      return {
        ...coworking,
        score: matchScore + eventsScore,
      };
    })
    .sort((a, b) => b.score - a.score);
}

(async () => {
  const sortedCoworkings = await sortCoworkingsByNetworkingPotential(
    userData,
    coworkingData
  );
  console.log(sortedCoworkings);
})();

async function getCoworkingRecommendations(userId) {
  const user = await User.findById(userId).lean();
  const coworkings = await Coworking.find().lean();

  // Preprocess user's skills
  let userSkills = [];
  for (const skill of user.skills) {
    userSkills.push(await lemmatize(skill));
  }

  // Content-Based Filtering: Find coworkings with relevant events
  let relevantCoworkings = [];
  for (const coworking of coworkings) {
    const events = await Event.find({ coworkingId: coworking._id }).lean();
    for (const event of events) {
      let eventTopics = [];
      for (const topic of event.topics) {
        eventTopics.push(await lemmatize(topic));
      }

      // If the event topics match with the user's skills, add the coworking
to the list
      if (eventTopics.some((topic) => userSkills.includes(topic))) {
        relevantCoworkings.push(coworking);
        break; // move to the next coworking
      }
```

```
    }
  }

  // Collaborative Filtering: Find coworkings visited by similar users
  const similarUsers = await User.find({ skills: { $in: userSkills }
}).lean();
  let similarUserCoworkings = [];
  for (const similarUser of similarUsers) {
    similarUserCoworkings.push(...similarUser.visitedCoworkings);
  }

  // Calculate scores for coworkings based on both relevance of events and
visits by similar users
  const coworkingScores = {};
  for (const coworking of relevantCoworkings) {
    const score = similarUserCoworkings.filter(
      (id) => id.toString() === coworking._id.toString()
    ).length;
    coworkingScores[coworking._id] = score;
  }

  // Sort coworkings by their scores in descending order
  const sortedCoworkings = relevantCoworkings.sort(
    (a, b) => coworkingScores[b._id] - coworkingScores[a._id]
  );

  return sortedCoworkings;
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ ТА МОНІТОРИНГУ РОБОТИ КОВОРКІНГІВ

Програма та методика тестування

КПІ.ІТ-9221.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Ірина ВІТКОВСЬКА

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Дмитро РУСАНОВСЬКИЙ

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є програмне забезпечення, що було розроблене в рамках даного проекту для автоматизації процесу управління та моніторингу роботи коворкінгів. Це програмне забезпечення включає в себе наступні основні складові:

– веб-клієнтську частину, яка є інтерфейсом для користувачів та адміністраторів. Вона включає в себе реалізацію реєстрації, авторизації, пошуку доступних коворкінгів, управління бронюваннями та декілька інших важливих функцій;

– серверну частину, що забезпечує логіку підбору доступних коворкінгів для користувача, а також контролює оновлення статусів бронювань;

– мобільні додатки для Android і iOS, які дозволяють користувачам коворкінгів використовувати всі функції системи зручно зі своїх мобільних пристроїв.

Кожна з цих складових вимагає окремого тестування, але вони також будуть випробовуватися разом, щоб переконатися в надійності взаємодії між різними компонентами системи.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка відповідності роботи програмного забезпечення вимогам, що були визначені в ході розробки. Це означає, що всі функції програмного забезпечення повинні працювати відповідно до їх опису;

– перевірка надійного зберігання та обробки даних. Програмне забезпечення повинно коректно обробляти всі вхідні дані, а також зберігати дані користувачів та коворкінгів в безпечному та цілісному стані;

– перевірка сумісності веб-додатку з останніми версіями основних браузерів (Google Chrome, Mozilla Firefox, Opera, Safari, Edge тощо);

– перевірка сумісності застосунку з різними операційними системами (Windows, macOS, Linux, Android, iOS);

– виявлення та усунення проблем, помилок і недоліків в програмному забезпеченні. Це може включати помилки в коді, проблеми з проектуванням інтерфейсу, а також проблеми з продуктивністю або безпекою;

– перевірка зручності і інтуїтивності користувацького інтерфейсу.

Програмне забезпечення повинно бути зручним для користувача, і всі його функції повинні бути легко зрозумілими і доступними.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– статичне тестування: цей метод полягає у перевірці програмного коду та відповідної документації на відповідність стандартам програмування, не запускаючи саму програму;

– динамічне тестування: цей метод застосовується в процесі виконання програми. Коректність роботи програмного забезпечення перевіряється на наборі тестових випадків, при цьому збираються та аналізуються дані про проблеми та помилки в роботі програми;

– функціональне тестування: перевіряється, чи відповідає реальна поведінка програмного забезпечення очікуваній на основі опису функцій в документації;

– системне тестування: перевіряється робота всього програмного забезпечення в цілому, звертаючи увагу на взаємодію окремих модулів та компонентів;

– мануальне тестування: це тестування, проведене людиною без використання автоматичних інструментів тестування. Тест-кейси розробляє та виконує тестувальник;

– тестування методом "чорної скриньки": об’єктом тестування є функції програми. Перевіряється коректність вихідних даних при заданих вхідних без врахування внутрішньої структури програми;

– тестування методом "білої скриньки": об’єктом тестування є внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним;

– тестування «сірої скриньки» – об’єктом тестування тут є деякі особливості внутрішньої поведінки програми. Перевіряється коректність вихідних даних при заданих вхідних, застосовується для тестування окремих алгоритму пошуку коворкінгів для користувача.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з використанням наскрізного тестування (End-to-end, E2E, Chain testing), з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

– динамічне тестування на відповідність функціональним вимогам;

– тестування на мобільних пристроях з різною роздільною здатністю екрану;

– тестування на мобільних пристроях з різною версією операційної системи;

– тестування на виведення повідомлень про помилку, коли це передбачено вимогами;

– тестування працездатності програми у випадку відсутності з’єднання до мережі;

– тестування інтерфейсу користувача;

– тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ ТА МОНІТОРИНГУ РОБОТИ КОВОРКІНГІВ

Керівництво користувача

КПІ.ІТ-9221.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Ірина ВІТКОВСЬКА

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Дмитро РУСАНОВСЬКИЙ

Київ – 2023

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«Co-work monitor» - це Progressive Web Application (PWA) та мобільний застосунок, розроблений з метою створення зручного середовища для пошуку та бронювання робочих місць в коворкінгах. Завдяки даному додатку, користувачі можуть швидко знайти вільні місця, забронювати їх, а також оплачувати онлайн, що значно спрощує процес організації роботи в коворкінгах. Також адміністраторам коворкінгів застосунок надає зручний інтерфейс для управління ресурсами коворкінгу, включаючи керування робочими місцями та їх бронюванням.

Кінцева збірка програмного забезпечення включає усі необхідні компоненти для його встановлення та використання, а саме: виконуваний файл для встановлення, бібліотеки та модулі, необхідні для роботи програми, та документацію користувача, що містить інструкції з встановлення та використання програмного забезпечення.

Інсталяційна версія програмного забезпечення передбачає простий та зручний процес встановлення, що включає всі необхідні компоненти для роботи програми. Користувачу просто потрібно запустити інсталяційний файл та слідувати інструкціям, які з'являться на екрані під час процесу встановлення.

Вміст репозиторію програмного забезпечення включає вихідний код, документацію розробника, вихідні файли для веб-версії клієнта, серверної частини, Android та iOS версій.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог окремо для PWA та мобільних додатків.

PWA версія:

– сучасний веб-браузер (Google Chrome, Firefox, Safari або Opera) останньої версії;

– наявність доступу до Інтернету;

– включений JavaScript у налаштуваннях браузера.

Мобільний додаток:

– мобільний пристрій з операційною системою Android 6.0 або вище, або iOS 10.0 або вище;

– вільне місце в пам'яті пристрою не менше 150 МБ для встановлення додатку;

– постійний доступ до Інтернету.

2.2 Завантаження застосунку

На даний момент застосунок можна встановити власноруч, у вигляді PWA та мобільного додатку для Android:

– PWA версія — перейдіть до веб-сайту "Co-work monitor" за адресою [https://diplom-b8df8.web.app]. Щоб встановити PWA, відкрийте веб-сайт в браузері свого пристрою, перейдіть до меню браузера і виберіть опцію "Додати до головного екрану". Після цього додаток буде встановлено на ваш пристрій і з'явиться на головному екрані;

– мобільний додаток для Android — завантажте APK-файл застосунку з [apkfab]. Перед встановленням файлу APK переконайтеся, що на вашому пристрої включено установки від невідомих джерел. Після завантаження відкрийте файл APK, щоб інсталювати додаток.

2.3 Перевірка коректної роботи

По завершенню встановлення додатка можна переконатися, що процес пройшов успішно, перевіривши наявність іконки додатка на головному екрані пристрою.

PWA версія — після встановлення PWA на ваш пристрій, іконка "Co-work monitor" має автоматично з'явитись на головному екрані вашого пристрою. Якщо іконка не з'явилась, спробуйте повторити процес встановлення. Після того, як ви відкриєте додаток, вас вітає початкова сторінка.

Мобільний додаток для Android — після завершення встановлення мобільного додатку, іконка "Co-work monitor" повинна з'явитись на головному екрані вашого Android-пристрою. Якщо ви не бачите іконки, переконайтеся, що ви дали дозвіл на встановлення додатків з невідомих джерел, і спробуйте встановити APK-файл знову. Коли ви відкриєте додаток, ви зможете побачити початкову сторінку додатка.

ВИКОНАННЯ ПРОГРАМИ

3.1 Виконання програми для користувача

При запуску PWA або мобільного застосунку є дві користувачу буде відображено сторінка авторизації (рисунок 3.1), що містить два поля – поле для введення електронної пошти і поле для паролю. Якщо вони будуть введені коректно, то після натискання кнопки “Sign in” користувач буде перенаправлений на сторінку пошуку коворкінгів.

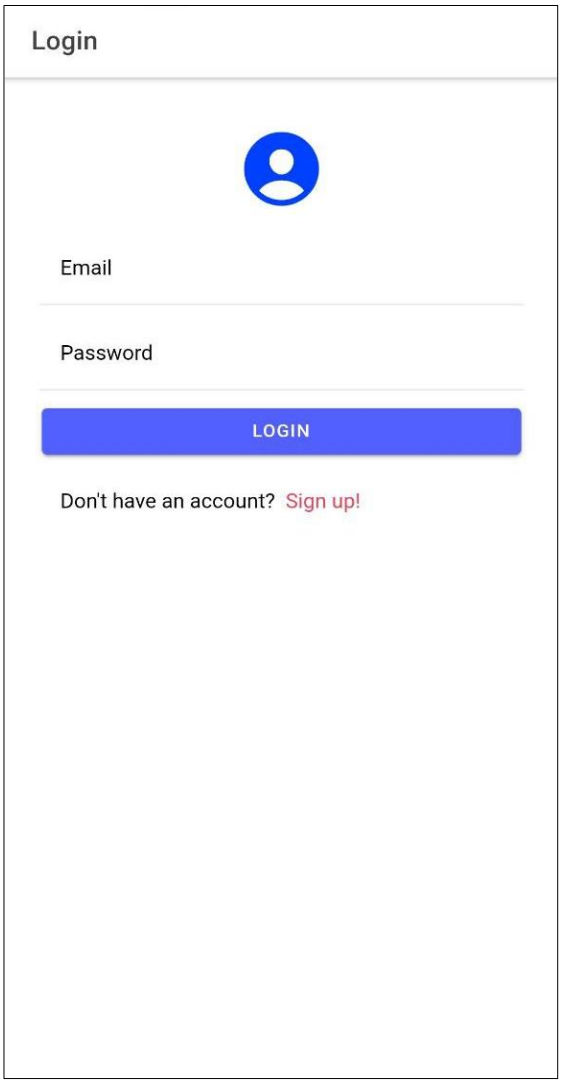


Рисунок 3.1 – Сторінка авторизації

Якщо користувач не зареєстрований, то для створення акаунту треба натиснути на кнопку “Sign up”. Це перенаправить користувача на сторінку

реєстрації (рисунок 3.2). Форма містить три поля: електронна пошта, пароль, форма для введення навичок користувача (по натисканню на кнопку-іконку плюса додається ще одне поле навичок). Електронна пошта повинна відповідати формату електронної пошти та не повинна використовуватись для реєстрації повторно. Пароль повинен бути від 6 символів у довжину. Якщо умови виконуються, то після натискання кнопки “Sign up” користувач буде перенаправлений на сторінку авторизації.

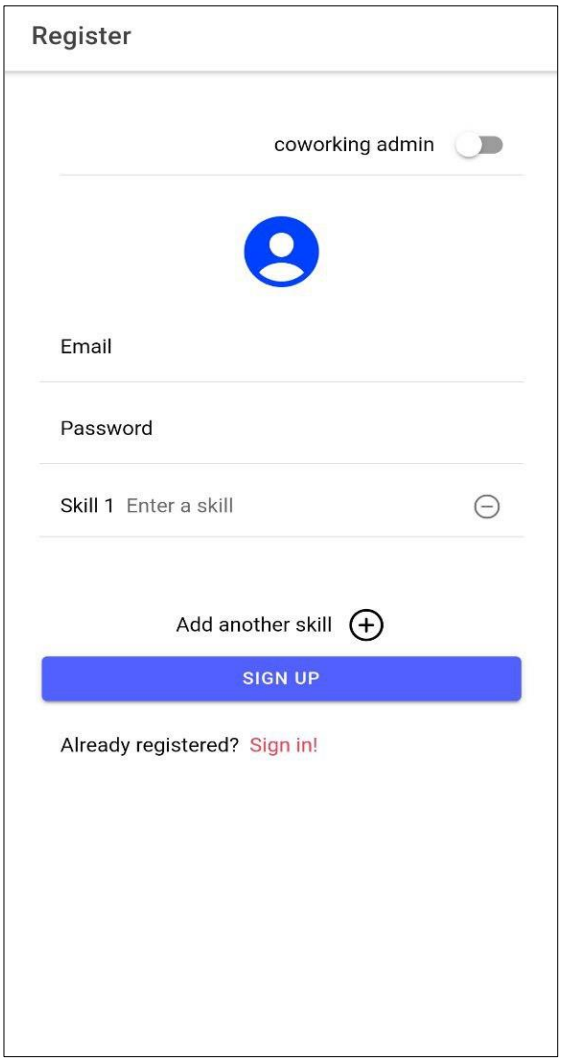


Рисунок 3.2 – Сторінка реєстрації

Після успішної реєстрації/ авторизації користувачу буде відображена сторінка пошуку коворкінгів (рисунок 3.3), на якій присутні поелементно у вигляді віртуального списку сутності коворкінгів з картинками та базовою

інформацією (назвою та локацією), а також кнопками “Book” для переходу на сторінку бронювання робочих місць в обраному коворкінгу.

В тулбарі сторінки знаходяться дві кнопки:

– з лівої сторони кнопка меню яка відкриває користувачу дві опції переходу на сторінки (перша — це сторінка на якій зараз знаходиться користувач, друга — це сторінка бронювань). На сторінці бронювань користувач бачить усі свої активні, скасовані та завершені бронювання. Користувач може скасувати актуальне бронювання (рисунок 3.4);

– з правої сторони кнопка кастомізації алгоритму пошуку коворкінгів, яка відриває модальне вікно з параметрами (локація користувача, діапазон цін та поля навичок користувача) після зміни параметрів та натискання на кнопку “Confirm” відбувається перезапуск алгоритму, який повертає нові результати пошуку коворкінгів (рисунок 3.5)

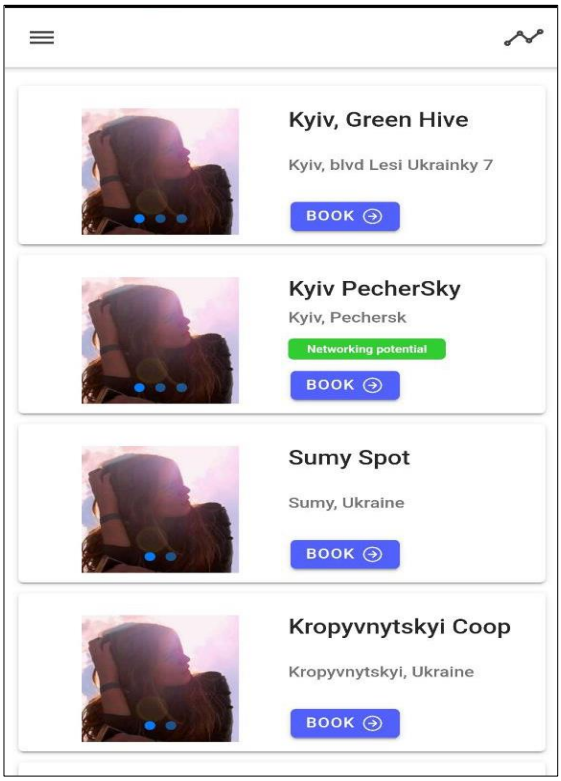


Рисунок 3.3 – Сторінка пошуку коворкінгів

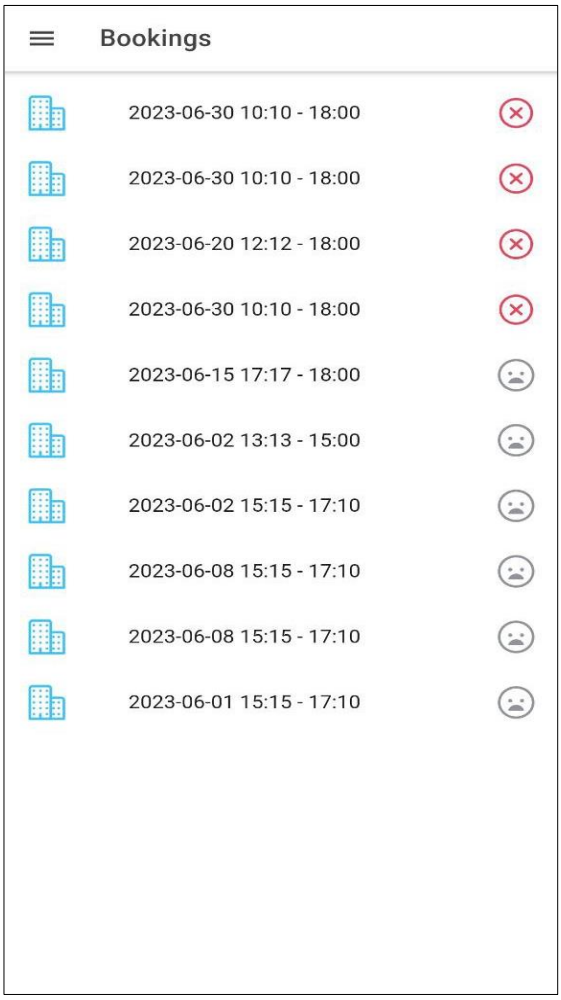


Рисунок 3.4 – Сторінка бронювань користувача

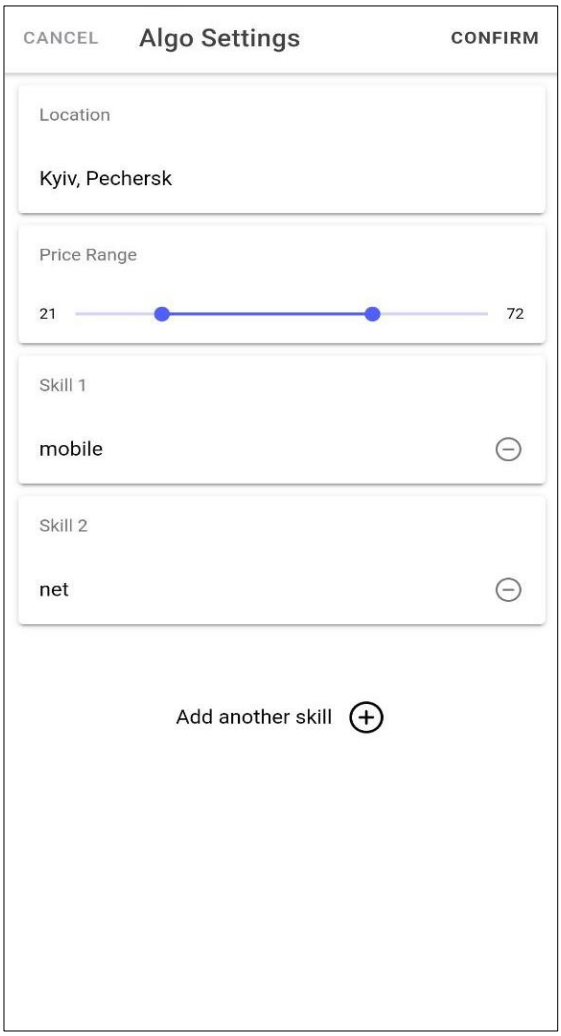


Рисунок 3.5 – Модальне вікно кастомізації алгоритму

Після того як користувач натискає на кнопку “Book” обраного зі списку коворкінгу, він переходить на сторінку бронювання робочих місць (рисунок 3.6), на якій будуть відображені фотографії коворкінгу та його детальна інформація (назва, локація та опис). На цій сторінці користувачу запропоновано календар з доступними для бронювання датами, після вибору дати користувачу має можливість вибору часового слоту та робочих місць на які можна здійснити бронювання. Після вибору робочого місця для бронювання користувач натискає на кнопку “Book”, таким чином створюючи актуальне бронювання, яке він може переглянути на сторінці своїх бронювань (рисунок 3.4)

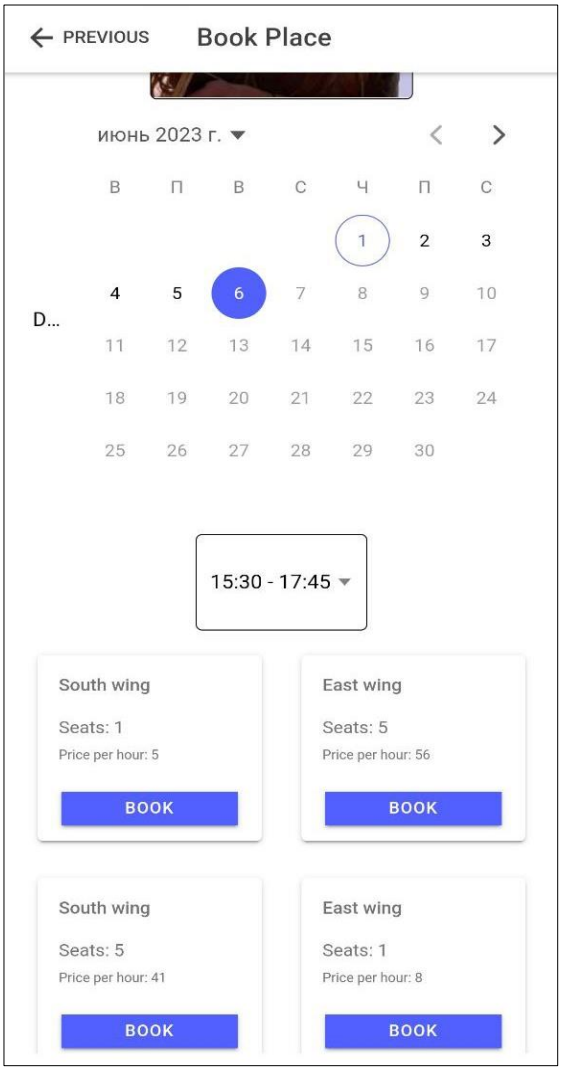


Рисунок 3.6 — Сторінка бронювання робочих місць в коворкінгу

3.2 Виконання програми для адміністратора коворкінгу

Для адміністраторів коворкінгів при запуску PWA або мобільного застосунку буде відображено сторінка авторизації, де йому так само як і користувачу потрібно ввести електронну адресу та пароль у відповідні поля на сторінці (рисунок 3.1), потім натиснути "Sign in".

Якщо адміністратор не зареєстрований на сторінці авторизації, внизу є кнопка "Sign up", яка перенаправляє на сторінку реєстрацію (рисунок 3.7).

Адміністратору коворкінгу потрібно перевести перемикач admin, зверху сторінки, в положення true. Далі потрібно ввести електронну адресу та пароль у відповідні поля, а потім натиснути "Sign up". Після цього адміністратор перенаправляється на сторінку реєстрації додавання інформації коворкінгу. Сторінку додавання коворкінгу з введеними даними (назва, локація, опис та посилання на картинки) вказано на рисунку 3.8

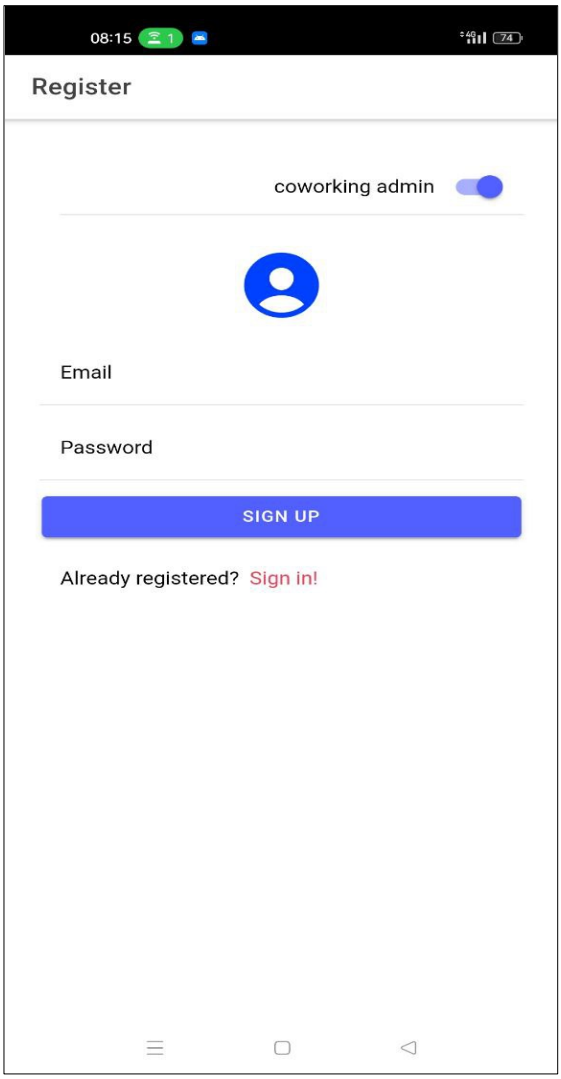


Рисунок 3.7 — Сторінка реєстрації (з перемикачем у стані admin = true)

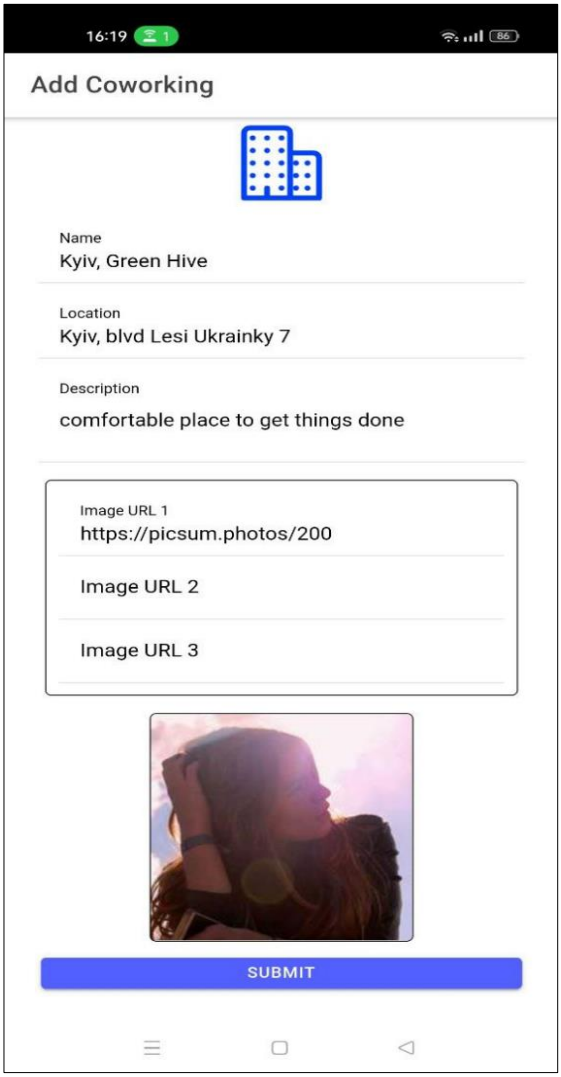


Рисунок 3.8 — Сторінка реєстрації коворкінгу з введеними даними

Після введення даних коворкінгу адміністратор натискає на кнопку Submit щоб підтвердити дані та зареєструвати коворкінг — перехід на сторінку робочих місць в своєму коворкінгу. При невідповідності даних поля будуть підсвічуватись красним кольором та натиснути кнопку “Submit” не вдасться. Також наступного разу коли адміністратор буде авторизуватись він одразу перейде на сторінку робочих місць коворкінгу.

Розглянемо приклад додавання робочого місця в коворкінг, адміністратор натискає на кнопку Add Place — відображається нова картка робочого місця, де від адміністратора очікується ввід інформації, приклад на рисунку 3.9.

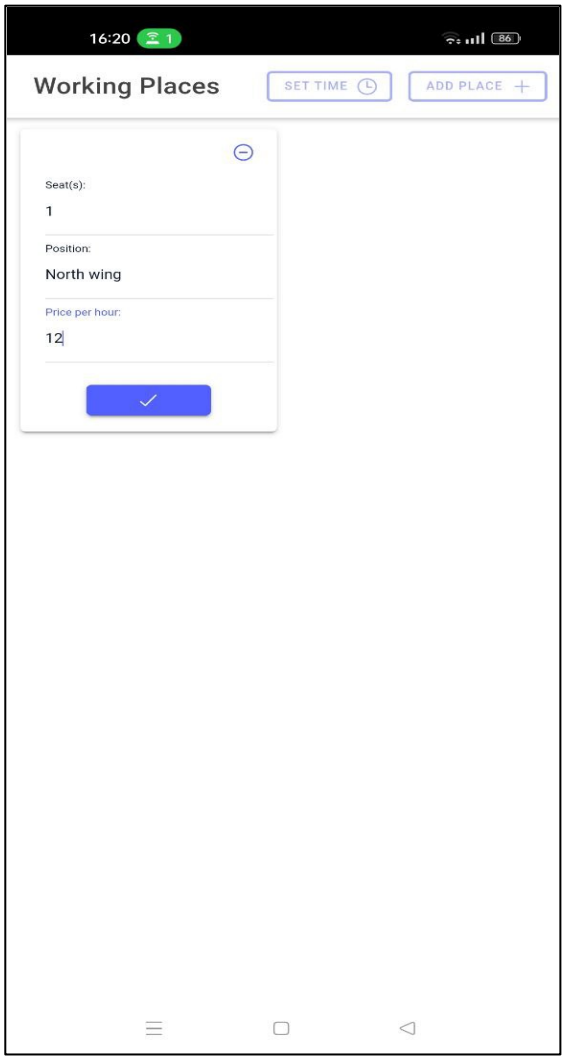


Рисунок 3.9 — Сторінка робочих місць коворкінгу

Після вказання інформації в відповідні поля картки робочого місця користувач натискає на кнопку галочки та робоче місце зберігається. Таким чином адміністратор може додати скільки завгодно робочих місць своєму коворкінгу. Потім для того щоб зробити ці місця доступними для бронювання користувачами застосунку — адміністратору потрібно виділити робочі місця та натиснути на кнопку “Set Time” по натисканню на цю кнопку у адміністратора буде дві опції функціоналу (які обираються перемикачем внизу сторінки) у відкритому модальному вікні (рисунок 3.10):

– вказати в системі стороннє бронювання робочого місця (місць);
– задати місцю (місцям) доступні часові слоти для бронювання.

Таким чином адміністратор додає свій коворкінг з робочими місцями доступними для бронювання у алгоритм підбору коворкінгу для користувачів.

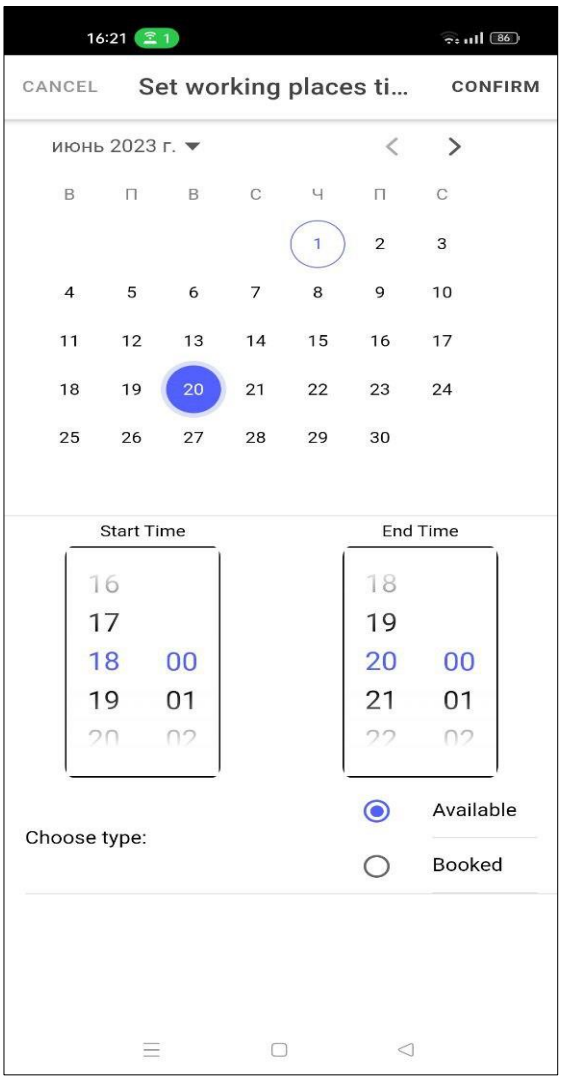


Рисунок 3.10 — Модальне вікно задавання доступного часу (бронювань) для робочих місць

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ ТА МОНІТОРИНГУ РОБОТИ КОВОРКІНГІВ

Графічний матеріал

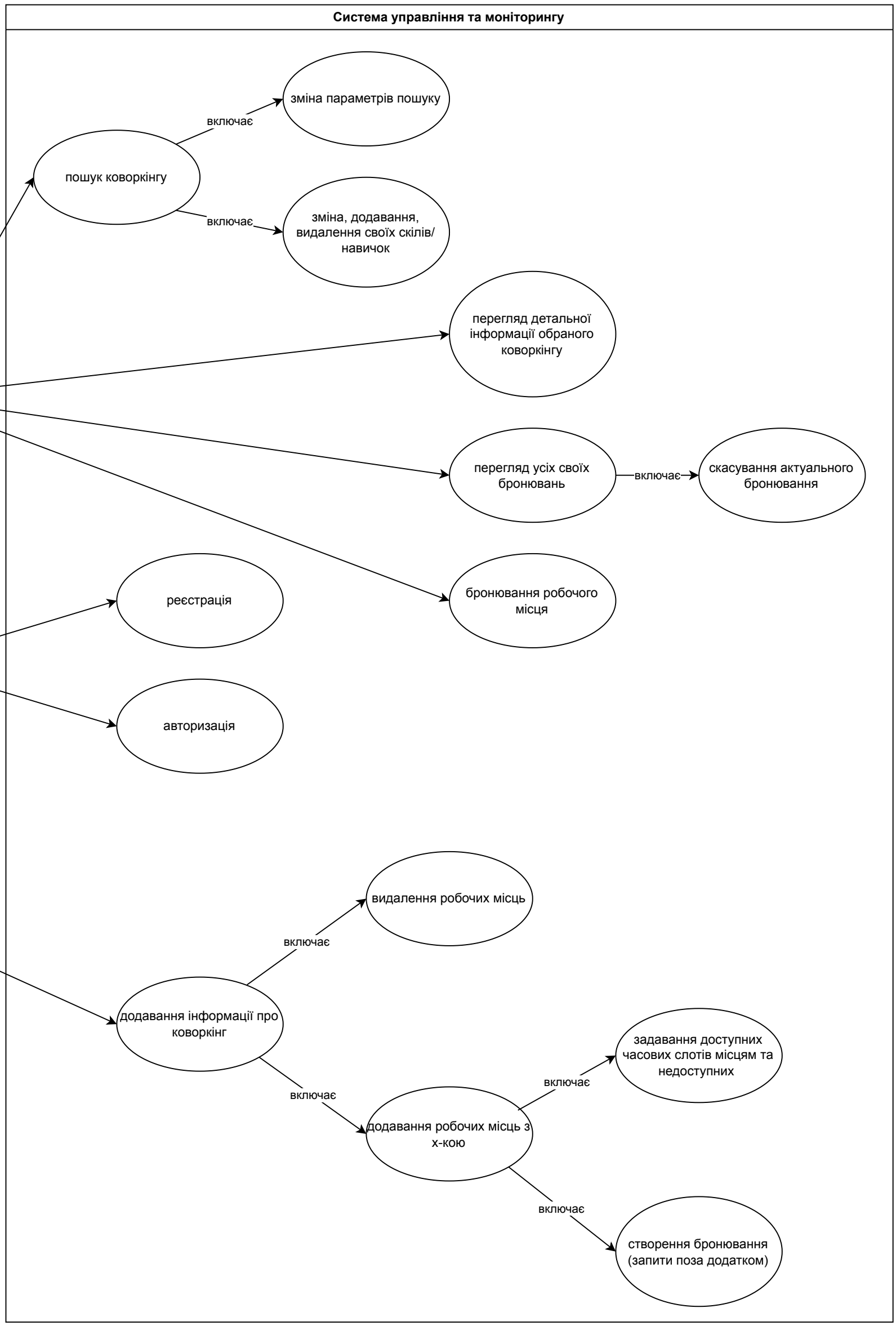
КПІ.ІТ-9221.045440.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Ірина ВІТКОВСЬКА

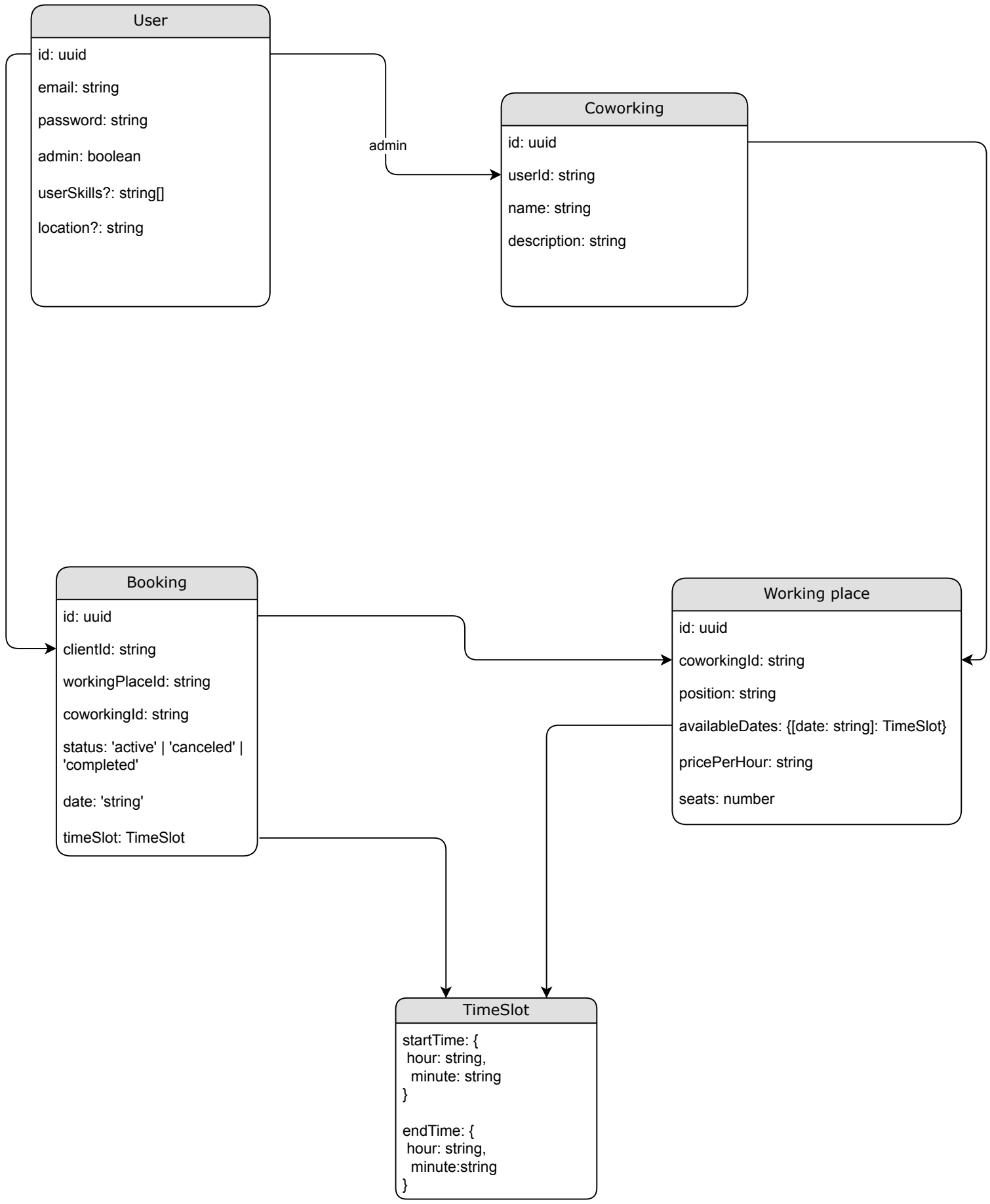
Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Дмитро РУСАНОВСЬКИЙ

Київ – 2023

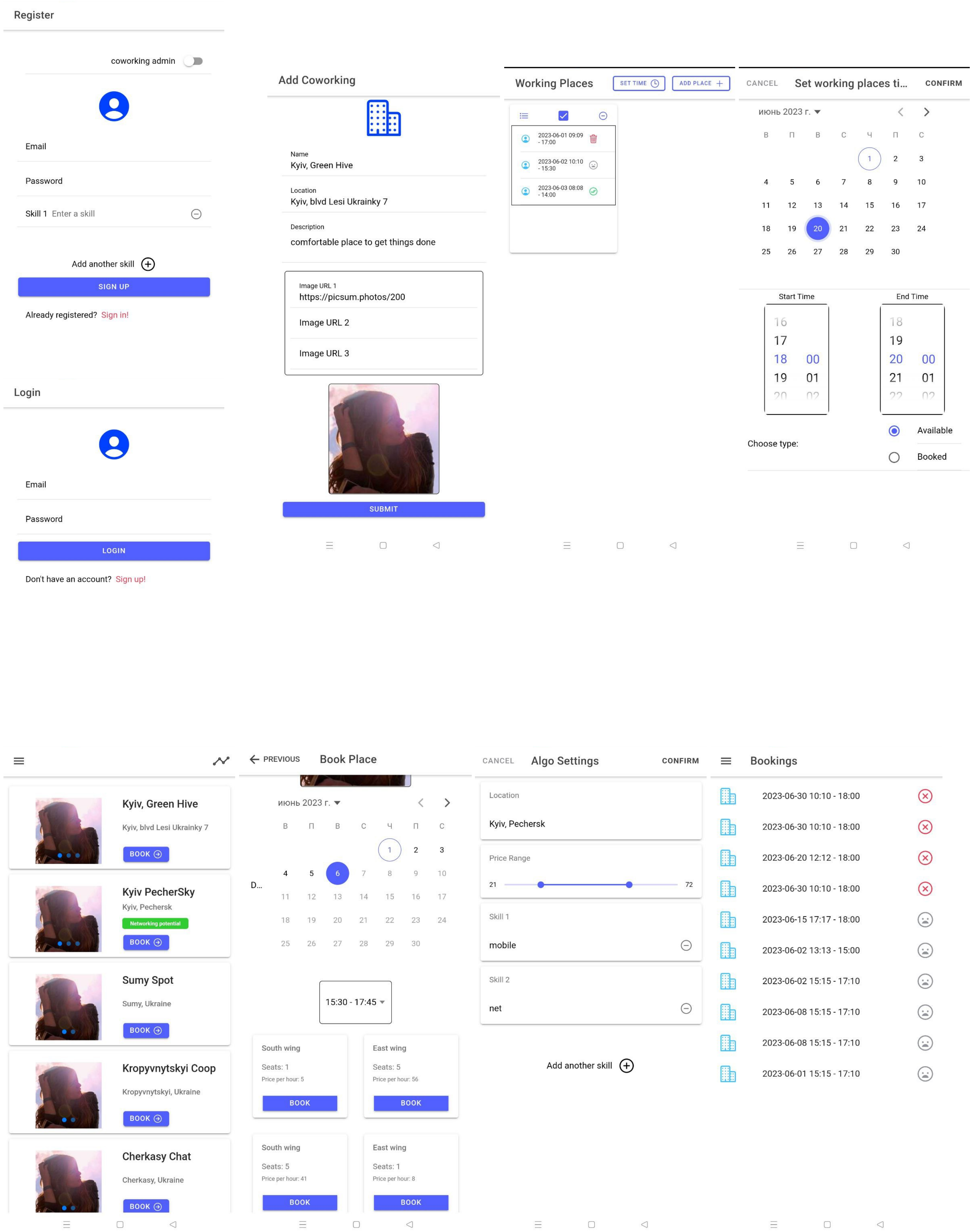


					КПІ.ІТ-9221.045440.06.99.ССВ
Зм.	Арк.	№ докум.	Підп.	Дата	
Розроб.		Русановський Д. Д.			Схема структурна варіантів використання
Перев.		Вітковська І. І.			Лит.: — Арк.: — Аркушів: 1
Т. Кон.					Аркуш — Аркушів
					Програмне забезпечення для управління та моніторингу роботи коворкінгів
Н. Кон.		Вітковська І. І.			КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-92
Затв.		Жаріков Е. В.			



					КПІ.ІТ-9221.045440.06.99.СБД	
Зм.	Арк.	№ докум.	Підп.	Дата		
Розроб.		Русановський Д. Д.			Схема бази даних	Лит. / Арк. / Аркушів: 1
Перев.		Вітковська І. І.				Аркуш / Аркушів
Т. Кон.					Програмне забезпечення для управління та моніторингу роботи коворкінгів	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-92
Н. Кон.		Вітковська І. І.				
Затв.		Жаріков Е. В.				

КПІ.ІТ-9221.045440.06.99.КЕ



Зм.	Арк.	№ докум.	Підп.	Дата
Розроб.		Русановський Д. Д.		
Перев.		Вітковська І. І.		
Т. Кон.				
Н. Кон.		Вітковська І. І.		
Зате.		Жаріков Е. В.		

КПІ.ІТ-9221.045440.06.99.КЕ

Креслення вигляду екранних форм

Лит.	Арк.	Аркушів
		1
Аркуш		Аркушів

Програмне забезпечення для управління та моніторингу роботи коворкінгів

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІТ-92