ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут телекомунікації

Кафедра системного аналізу

Пояснювальна записка

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «**Інформаційна автоматизована система резервації робочих місць для співробітників компанії**»

Виконав: студент 4 курсу,
групи САД–41, спеціальності
124 Системний аналіз
(шифр і назва спеціальності)
Олійник А.С.
(прізвище та ініціали)
Керівник Холоднюк С.З.
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Телекомунікацій

Кафедра Системного аналізу

Ступінь вищої освіти «Бакалавр»

Спеціальність 124 Системний аналіз

ЗАТВЕРДЖУЮ

Завідувач кафедри

Системного аналізу

Гордієнко Т.Б

(ПІБ)

Дата: _____ _____ 2023 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Олійнику Артемію Сергійовичу

(прізвище, ім‘я, по батькові)

1. Тема роботи: Інформаційна автоматизована система резервації робочих місць для співробітників компанії

Керівник роботи Холоднюк Сергій Зеновійович, к.е.н.

(прізвище, ім‘я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від _____ 2023 року № .

2. Строк подання студентом роботи _____

3. Вхідні дані до роботи:

_____

_____

_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

_____

_____

5. Перелік графічного матеріалу

1) _____

2) _____

3) Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1			
2			
3			
4			
5			
6			
7			
8			
9			

Студент _____ Олійник А.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Холоднюк С.З.
(підпис) (прізвище та ініціали)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

ПОДАННЯ

ГОЛОВІ ДЕРЖАВНОЇ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ

ЩОДО ЗАХИСТУ БАКАЛАВРСЬКОЇ РОБОТИ

Направляється студент Олійник А.С. до захисту бакалаврської роботи
(прізвище та ініціали)

за спеціальністю 124 Системний аналіз
(шифр і назва спеціальності)

на тему: «Інформаційна система управління проектами»

Бакалаврська робота і рецензія додаються.

Директор інституту _____ В.І. Кравченко
(підпис)

Довідка про успішність

Олійник А.С. за період навчання в Навчально-науковому інституті Телекомунікації
(прізвище та ініціали студента)

з 2021 року до 2023 року повністю виконав навчальний план за спеціальністю з таким розподілом оцінок за:

національною шкалою: відмінно _____%; добре _____%; задовільно _____%;

шкалою ECTS: A_____%; B_____%; C_____%; D_____%; E_____%.

Методист факультету: _____ _____
(підпис) (прізвище та ініціали)

Висновок керівника бакалаврської роботи

Студент Олійник А.С.

_____

_____

_____

_____

Все це дозволяє оцінити виконану бакалаврську роботу студента Костенка В.К. на оцінку «_____» та присвоїти йому кваліфікацію фахівець з інформаційних технологій.

Керівник роботи _____ _____
_____
(підпис) (прізвище та ініціали)

_____ _____ 2023 року

Висновок кафедри про бакалаврську роботу

Бакалаврську роботу розглянуто(а). Студент Олійник А.С. допускається до захисту даної
(прізвище та ініціали)
роботи в Державній екзаменаційній комісії.

Завідувач кафедри Системного аналізу _____ Гордієнко Т.Б.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 79 с., 73 рис., 18 джерела.

Об’єкт дослідження –

Предмет дослідження –

Основною ціллю роботи є розробка та впровадження ефективної системи, яка допоможе оптимізувати процес резервації робочих місць для співробітників компанії.

Основна мета цієї системи полягає в полегшенні та автоматизації процесу резервації робочих місць, забезпеченні максимальної ефективності використання робочих місць та задоволення потреб співробітників щодо робочого простору. У роботі вивчаються різні існуючі рішення, які використовуються в практиці для автоматизації процесу резервації робочих місць в компаніях. Ці рішення включають різні програмні та технічні рішення, а також методики та підходи до організації системи резервації робочих місць.

У роботі пропонується альтернативне рішення, яке базується на використанні інтерактивної карти резервації робочих місць. Система дозволяє співробітникам компанії в режимі онлайн переглядати доступні робочі місця на інтерактивній карті офісу, обирати зручне місце, а також здійснювати всі взаємодії з додатком безпосередньо з мобільних пристроїв.

На основі результатів виконаних досліджень розроблено алгоритми, які застосовуються для процесу бронювання робочого місця.

ЗМІСТ

ВСТУП **9**
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ **11**
1.1. Опис завдання 11
1. Реєстрація. 11
2. Авторизація. 12
3. Створення компанії. 13
4. Створення офісу. 13
5. Створення поверху. 14
6. Перегляд співробітників в офісі. 15
7. Додавання співробітника в офіс 15
8. Інтерактивна карта поверху зі столами. 15
1.2. Огляд існуючих рішень 16
1. DeskFlex 16
2. Roomzilla 16
3. Robin 17
4. Skedda 17
5. Teem 17
Висновки до розділу 1 18
РОЗДІЛ 2 ПІДГОТОВКА ДО РОЗРОБКИ ПЗ **19**
2.1. Огляд популярних мов програмування для серверної частини 19
2.2. Огляд популярних систем управління базами даних 21
2.3. Огляд популярних мов програмування для клієнтської частини 32
2.3. Огляд допоміжних технологій та фреймворків 35
Висновки до розділу 2 36
РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ **37**
3.1. Архітектура додатків 38
3.2. Архітектура серверної частини програмного продукту 40
3.3. Архітектура клієнтської частини програмного продукту 50
Висновки до розділу 3 69
РОЗДІЛ 4 МОДЕЛЮВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ **70**
4.1. Взаємодія системи з користувачем 71
Висновки до розділу 75
ВИСНОВКИ **76**
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ **77**

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення;

ORM - об’єктно-реляційна модель;

СУБД - система управління базами даних;

HTTP(Hypertext Transfer Protocol) - протокол передачі даних, на основі якого працює всесвітня павутина

ВСТУП

Інформаційні технології знаходять своє застосування в різних сферах життя, від роботи до особистих потреб людей. Швидкий розвиток цієї галузі дає людям нові можливості полегшити життя та покращити продуктивність роботи. Наприклад, використання комп’ютерів дозволяє виконувати складні обчислення швидко та ефективно, що важко було б зробити вручну. Крім того, завдяки інформаційним технологіям людина може спілкуватися з іншими людьми незалежно від їх місця знаходження, отримувати доступ до необхідної інформації та послуг, що заощаджує час та кошти.

Звичайно, постійне використання інформаційних технологій може мати й негативний вплив на життя людей, наприклад, призвести до зменшення «живої» комунікації та може погіршити здоров'я через постійний сидячий спосіб життя. Проте, якщо використовувати інформаційні технології з користю, вони можуть надавати багато інших можливостей, які полегшують життя та покращують продуктивність роботи.

З розвитком інформаційних технологій все більше і більше послуг можна отримати онлайн, що дозволяє здійснювати покупки, звертатися до банківських послуг, спілкуватися з друзями та колегами через соціальні мережі, знаходити потрібну інформацію та вчитися за допомогою дистанційного навчання. Всі ці можливості дозволяють людям зекономити час та зробити своє життя більш комфортним та продуктивним.

Таким чином, актуальність розробки інформаційної автоматизованої системи резервації робочих місць для співробітників компанії полягає у необхідності покращення умов праці, зниженні затримок у виконанні завдань та підвищенні продуктивності працівників. Проблема нестачі вільних робочих місць досить поширена в багатьох компаніях, тому ця тема є досить вивченою. Однак, існує потреба у розробці інформаційної системи, яка б дозволяла автоматизувати процес

резервації робочих місць та допомагала ефективно використовувати ресурси підприємства.

Об'єктом дослідження є процес резервації робочих місць для співробітників компанії, а предметом дослідження є розробка інформаційної автоматизованої системи, яка забезпечує можливість ефективного управління резервацією робочих місць та забезпечення комфортних умов праці для співробітників.

Метою даного дослідження є розробка інформаційної системи резервації робочих місць для співробітників компанії з метою поліпшення умов праці, зниження затримок у виконанні завдань та підвищення продуктивності працівників.

Результатом даного дослідження є розробка та імплементація спеціальної системи, яка дозволяє забронювати робоче місце за допомогою інтерактивної карти. Це дозволяє працівникам зручно обирати місце для роботи, переглядаючи доступні варіанти на карті, та резервувати його. Така система може забезпечити ефективне використання простору та зменшити конфлікти між працівниками щодо використання робочих місць. Також ця система може бути корисною для менеджменту компанії, який може відстежувати загальну використаність робочого простору та регулювати кількість доступних місць у залежності від потреб.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Опис завдання

Метою даного дипломного проекту є розробка інформаційної автоматизованої системи резервації робочих місць для співробітників компанії. Система повинна бути доступна на мобільних платформах Android та iOS. Основною функцією системи буде забезпечення співробітників компанії можливістю зарезервувати робоче місце в офісі заздалегідь, що дозволить уникнути зайнятості всіх місць та забезпечити оптимальне використання простору.

Система буде складатися з наступних основних сторінок:

1. Реєстрація;
2. Авторизація;
3. Створення компанії;
4. Створення офісу;
5. Створення поверху;
6. Перегляд співробітників в офісі;
7. Додавання співробітника в офіс;
8. Інтерактивна карта зі столами.

1.Реєстрація.

Сторінка, на якій співробітники компанії можуть створити свій акаунт в системі. Для реєстрації користувач повинен ввести свої персональні дані, такі як електронну пошту та пароль. Ім'я та прізвище - опціональні. Аватар з ініціалами або першою буквою електронної пошти змінюється після зміни одного з трьох значень: електронна пошта, ім'я або прізвище.

Також можуть бути додаткові поля для введення інформації, якщо це необхідно для компанії, наприклад, номер телефону або посада в компанії. Після

заповнення всіх полів користувач натисне кнопку "Зареєструватися" (“Sign up”), щоб створити свій акаунт.

При реєстрації в системі може бути встановлений механізм підтвердження акаунта, наприклад, через електронну пошту або СМС. Це допоможе підвищити безпеку системи та перевірити, що реєстрацію проводить сам співробітник, а не хтось інший.

Крім того, на цій сторінці будуть встановлені правила для створення безпечного пароля, щоб підвищити захист акаунта. Для проходження перевірки, пароль повинен бути довжиною від 5 до 256 символів. Також необхідно перевірити правильність введення електронної пошти користувача, та вивести повідомлення про помилку у разі не відповідності введених даних.

Після успішної реєстрації користувач буде перенаправлений на сторінку авторизації, де він зможе увійти в свій акаунт та отримати доступ до функціоналу системи.

2. Авторизація.

Сторінка авторизації, дозволить зареєстрованим користувачам увійти в систему та отримати доступ до резервування робочих місць. На цій сторінці буде розміщена форма, де користувач буде повинен ввести свою електронну пошту та пароль, які він вказував при реєстрації в системі. Після введення правильних даних, якщо користувач не є адміністратором, система перенаправить його на сторінку для резервування робочого місця. У разі некоректних даних, на сторінці буде показане повідомлення про помилку, та користувач повинен ввести правильні дані для входу в систему.

Якщо ж поточний користувач є адміністратором системи, то він буде перенаправлений на одну з 4 сторінок, в залежності від того, що потрібно створити для правильної роботи системи:

1. Створення компанії;
2. Створення офісу;
3. Створення поверху;

4. Карта поверху.

3.Створення компанії.

На сторінці створення компанії адміністратор компанії буде мати можливість створити нову компанію в системі. На цій сторінці буде доступне поле для введення назви компанії та кнопка “Створити компанію” (“Create company”), після натискання якої запит на створення компанії буде відправленний.

Після створення компанії, адміністратор зможе додавати офіси до компанії. Якщо адміністратор закриє додаток і відкриє його знову, та авторизується, система автоматично впізнає, чи була створена компанія. У разі, якщо компанія не була створена, користувач буде перенаправлений на сторінку створення компанії. Якщо компанія вже була створена, користувач буде перенаправлений на сторінку створення офісу.

Створення компанії є важливим етапом для подальшої роботи з системою резервації робочих місць. Воно забезпечує можливість групування співробітників та робочих місць відповідно до структури компанії. Це дозволить забезпечити зручний та легкий доступ до інформації про резервування місць для кожного з співробітників та оптимальне використання простору офісу.

4.Створення офісу.

На сторінці "Створення офісу" (“Create office”) адміністратор компанії зможе створити новий офіс і додати до нього користувачів. При створенні офісу адміністратор повинен ввести назву офісу в текстове поле. Після цього, адміністратор може натиснути кнопку "Додати користувачів" (“Add users”), яка перенаправить користувача на сторінку перегляду користувачів офісу. Також на цій сторінці повинна бути присутня кнопка “Створити офіс” (“Create office”), після натискання якої запит на створення офісу буде відправленний.

Після створення офісу, адміністратор зможе додавати поверхи до офісу, де співробітники зможуть резервувати робочі місця. Якщо адміністратор закриє додаток і відкриє його знову, та авторизується, система автоматично впізнає, чи був створений офіс. У разі, якщо офіс не був створений, користувач буде

перенаправлений на сторінку створення офісу. Якщо офіс вже був створений, користувач буде перенаправлений на сторінку створення поверху.

5.Створення поверху.

На сторінці "Створення поверху" (“Create floor”) адміністратор зможе ввести номер поверху та завантажити картинку плану поверху. Для цього на сторінці буде розміщено поле для вводу номеру поверху та спеціальна кнопка для завантаження картинки поверху. Після завантаження картинки, вона буде відображена на сторінці. Також на сторінці буде кнопка "Створити поверх" (“Create floor”), після натискання якої буде відправлений запит на створення поверху в системі.

Після створення поверху, адміністратор матиме можливість додавати, видаляти та змінювати положення столів, а також повертати їх. У випадку, якщо адміністратор закриє додаток і знову його відкриє, після авторизації система автоматично перевірить, чи був створений поверх. Якщо поверх ще не був створений, користувач буде перенаправлений на сторінку створення поверху. Якщо ж поверх вже був створений, користувач буде перенаправлений на інтерактивну карту поверху.

6.Перегляд співробітників в офісі.

Цей пункт передбачає розробку сторінки, на якій адміністратор офісу зможе переглянути список співробітників, які мають доступ до офісу. На цій сторінці буде доступна інформація про роль, ім'я, прізвище та електронну пошту співробітника. Крім того, на сторінці будуть розташовані кнопки для додавання нового користувача та повернення до попередньої сторінки. Таким чином, адміністратор зможе ефективно керувати доступом співробітників до офісу та контролювати розподіл місць роботи.

7.Додавання співробітника в офіс.

Сторінка буде містити функціонал, що дозволяє адміністратору додати нового співробітника в систему. Для цього буде створена окрема сторінка, яка буде схожа на сторінку реєстрації. На цій сторінці буде розміщена форма, в якій адміністратор повинен ввести ім'я, прізвище, роль, електронну пошту та пароль для

нового співробітника. На данний момент в системі доступні 2 ролі: співробітник (employee), адміністратор (office manager).

Після заповнення форми та натискання кнопки "Додати співробітника" (“Add user”), дані будуть перевірені на відповідність вимогам та, якщо дані коректні, новий співробітник буде доданий до бази даних системи.

Ця функція буде доступна лише адміністраторам системи та забезпечить швидке та зручне додавання нових співробітників до системи резервації робочих місць.

8.Інтерактивна карта поверху зі столами.

Інтерактивна карта поверху зі столами буде однією з ключових функцій інформаційної системи. На цій карті будуть відображені всі столи на кожному поверсі офісу. Користувачі зможуть зарезервувати вільні столи, просто клікнувши на відповідний стіл на карті.

Адміністратор системи матиме можливість додавати, видаляти та змінювати положення столів на карті. Він також зможе повертати стіл на 360 градусів, якщо це буде необхідно для оптимального використання простору. Це дозволить адміністратору легко адаптувати мапу до будь-яких змін у просторі офісу та забезпечити актуальну інформацію для користувачів системи.

1.2. Огляд існуючих рішень

Управління офісним простором може бути складною задачею, особливо у великих компаніях з багатьма співробітниками та різними відділами. Щоб забезпечити ефективну роботу та максимальне використання офісного простору, існують різноманітні додатки для менеджменту офісу. У цьому пункті будуть розглянуті деякі з найбільш популярних інструментів.

1. DeskFlex

DeskFlex - це додаток для резервації робочих місць та зберігання даних про використання офісного простору. З його допомогою співробітники можуть забронювати необхідні робочі місця та переговорні кімнати, а також відстежувати

доступність цих місць в режимі реального часу. DeskFlex також надає звіти про використання простору та може бути інтегрований з іншими корпоративними системами.

2. Roomzilla

Roomzilla - це додаток для резервації переговорних кімнат, який дозволяє забронювати кімнати в режимі реального часу та стежити за їх доступністю. Roomzilla також надає можливість змінювати конфігурацію кімнат та налаштовувати нагадування для співробітників про зближення часу зустрічі. Додаток може бути інтегрований з календарями Google та Outlook.

3. Robin

Robin - це додаток для резервації робочих місць, переговорних кімнат та інших приміщень у компанії. Robin дозволяє співробітникам бронювати місця та налаштовувати нагадування про зустрічі. Крім того, додаток надає звіти про використання офісного простору та може бути інтегрований з календарями Google та Outlook. Robin також має можливість стеження за використанням приміщень та аналізу даних про використання простору.

4. Skedda

Skedda - це додаток для резервації робочих місць та переговорних кімнат у компанії. Skedda надає можливість налаштовувати розклади робочих місць та кімнат, а також відстежувати доступність цих місць у режимі реального часу. Додаток може бути інтегрований з іншими системами управління спорудами та корпоративними системами.

5. Teem

Teem - це додаток для управління офісним простором, який дозволяє співробітникам бронювати робочі місця та переговорні кімнати. Teem надає звіти про використання простору та може бути інтегрований з іншими корпоративними системами, такими як Google Apps та Slack. Додаток також має можливість стеження за використанням приміщень та аналізу даних про використання простору.

Одним з головних недоліків всіх розглянутих рішень є те, що вони є платними. Це може бути значним обмеженням для невеликих компаній або стартапів, які можуть не мати достатньої фінансової потужності для оплати подібного програмного забезпечення.

Тому, створення системи з відкритим вихідним кодом може бути головною перевагою дипломного проекту. Це дасть змогу будь-якій компанії скористатись системою безкоштовно та при необхідності змінювати її функціонал під свої потреби.

Програмне забезпечення з відкритим вихідним кодом дозволяє зробити розробку більш доступною та привабливою для потенційних користувачів, збільшує кількість можливих внесків до проекту з боку програмістів, що робить проект більш стійким та масштабованим.

Ще одною перевагою відкритого вихідного коду є можливість безкоштовного вдосконалення системи через внесення змін від програмістів, які є зацікавленими у вдосконаленні проекту.

Висновки до розділу 1

У розділі 1 було проаналізовано предметну область, визначено основні вимоги для проєкту.

Було проведено огляд та аналіз існуючих рішень з точки зору функціональності кожного з них, визначено основний недолік кожного розглянутого аналога.

Створення системи з відкритим вихідним кодом є головною перевагою дипломного проекту, оскільки вона може стати доступнішою та привабливішою для потенційних користувачів та сприяти подальшому розвитку системи шляхом залучення внесків від програмістів.

РОЗДІЛ 2
ПІДГОТОВКА ДО РОЗРОБКИ ПЗ

Як вже було зазначено в темі проекту та попередньому розділі, програмне забезпечення, яке є результатом цього проекту – це додаток, який буде доступний на Android та iOS. Тому необхідно розглянути технології, які є популярними у цей час для розробки програмного забезпечення подібного призначення. У цьому розділі будуть розглянуті :

- популярні мови програмування та визначено мову програмування, якою буде написана програмна реалізація проекту;
- відомі системи управління базами даних;
- технології для розробки клієнтської частини;
- допоміжні технології та фреймворки;
- середовища розробки програм для обраної мови програмування.

2.1. Огляд популярних мов програмування для серверної частини

Задача серверної частини полягає у забезпеченні безперебійної роботи інформаційної системи, що потребує використання високопродуктивних технологій програмування. Основні критерії вибору мов програмування для серверної частини - це продуктивність, масштабованість, надійність та зручність для розробників.

1. Java [1] є дуже популярною мовою програмування, яка застосовується більш ніж у 90% найбільших компаній у світі, незважаючи на те, що вона була створена ще у 1995 році. Причиною такої популярності є використання Java для розробки мобільних додатків на платформі Android, яка є найпопулярнішою мобільною платформою. Крім того, Java є кросплатформною мовою, що дозволяє її використання для розробки десктопних додатків та бек-енд частин програмного забезпечення.

2. Мова програмування Python [2]. Основною перевагою цієї мови є те, що мова Python є досить простою для вивчення, а код на мові Python досить легко

читається, що і вплинуло на те, що ця мова є однією з найпоширеніших на сьогоднішній день. Python також є кросплатформним. Має потужні інструменти для роботи з базами даних, операційною системою тощо. Досить часто застосовується для реалізації ігор або вебдодатків, є досить популярним у наукових обчисленнях та дослідженнях. Водночас недоліком мови Python може бути не найкраща швидкість виконання.

3. JavaScript [3] є ще однією популярною мовою програмування для серверної частини. JavaScript має великий потенціал для розробки високоякісних веб-додатків, а також забезпечує швидку розробку завдяки наявності великої кількості фреймворків та бібліотек.

4. Мова програмування Go [4] стала популярною для серверної розробки завдяки своїй продуктивності та здатності працювати з великими об'ємами даних. Крім того, Go є досить простою мовою програмування, що робить її зручною для розробників з різним рівнем досвіду.

5. Мова програмування C++ [5]. Це мова програмування, яка підтримує об'єктно-орієнтовану та процедурну парадигми програмування, має потужні бібліотеки для розробки. Надає програмісту можливість працювати з пам'яттю на більш низькому рівні, що надає перевагу C++ над іншими мовами програмування у швидкості виконання програм, однак в той же час вимагає від програміста вищого рівня кваліфікації, ніж інші мови програмування. Ця мова програмування чудово підходить для проектів, де важливою складовою є продуктивність та швидкість роботи. Основними напрямками використання C++ є оптимізація програм, написаних на інших мовах програмування з метою покращення швидкодії. Також є популярною мовою для розробки ігор, особливо тих, де потрібна оптимізація тощо.

6. Інші популярні мови програмування для серверної частини включають Ruby, PHP, C#, Rust та інші. Вибір мови програмування для серверної частини залежить від конкретних потреб проекту та вимог до продуктивності, масштабованості та надійності системи.

З усіх мов програмування, які були згадані, для написання серверної частини прокту була обрана Java. Є декілька причин, чому ця мова є найкращим вибором для цього проекту.

По-перше, Java має велику спільноту розробників та широкий спектр фреймворків та бібліотек для веб-розробки. Це дозволяє розробникам створювати високоякісні веб-додатки швидко та ефективно. Java також має добре розроблену систему безпеки, що робить її ідеальним вибором для веб-додатків, які мають обмінюватися чутливою інформацією.

По-друге, Java має велику продуктивність та масштабованість. Java відома своєю здатністю працювати з великими обсягами даних та забезпечувати високу продуктивність. Java також дозволяє легко масштабувати серверну частину, що робить її ідеальним вибором для веб-додатків, які потребують багато ресурсів.

По-третє, Java є дуже надійною мовою програмування. Її використовують багато великих корпорацій та урядових установ, що свідчить про її надійність та стійкість. Це важливо для серверної частини, яка повинна працювати без збоїв та збоїв.

Отже, на основі цих факторів можна зробити висновок, що Java є найкращою мовою програмування для реалізації серверної частини інформаційної автоматизованої системи резервації робочих місць для співробітників компанії.

2.2. Огляд популярних систем управління базами даних

Для того, щоб провести огляд систем управління базами даних, необхідно визначити це поняття згідно термінології. Отже, система управління базами даних (СУБД) — набір взаємопов'язаних даних і певна множина програм для доступу до цих даних [3].

Також необхідно розглянути класифікацію СУБД за моделлю даних. За цим параметром найчастіше виділяють наступні види :

Ієрархічна модель - це одна з найстаріших моделей даних, яка застосовувалась у реляційних базах даних в минулому. Ця модель даних базується

на деревоподібній структурі, яка нагадує генеалогічне дерево. Особливістю ієрархічної моделі є те, що для кожного головного (батьківського) об'єкта може існувати кілька підлеглих (дочірніх), а для кожного підлеглого об'єкта може бути тільки один головний.

У ієрархічній моделі даних можливі лише два типи зв'язків між об'єктами: "один до одного" і "один до декількох". Це означає, що кожен підлеглий об'єкт може мати тільки одного батьківський об'єкт, але батьківський об'єкт може мати декілька підлеглих об'єктів.

Недоліком ієрархічної моделі даних є її негнучкість. Оскільки в моделі не передбачено можливість наявності у об'єкта декількох "батьків", вона не є придатною для зберігання інформації, що містить багато-багато зв'язків між різними об'єктами. Також відсутність прямого доступу до даних у такій моделі даних робить її непридатною в умовах частого виконання запитів, які не є запланованими заздалегідь.

Іншою недолікою ієрархічної моделі даних є її складність. Оскільки в моделі даних має бути врахована ієрархія між об'єктами, її створення і підтримка може бути дуже часомісткою та складною задачею.

Незважаючи на свої недоліки, ієрархічна модель даних все ще застосовується в деяких сферах, де ієрархічна структура є невід'ємною частиною даних. Наприклад, в галузі генеалогії ієрархічна модель даних може бути досить ефективною, оскільки вона дозволяє організувати інформацію про сімейні зв'язки.

Однак, у більшості випадків використання ієрархічної моделі даних обмежується, оскільки існують інші більш гнучкі та ефективні моделі даних, такі як реляційна модель даних або об'єктно-орієнтована модель даних. У цих моделях даних більше можливостей для зберігання складних зв'язків між різними об'єктами та прямого доступу до даних.

Мережева модель є однією з основних моделей даних, що знаходить широке застосування в сучасному програмуванні та інформаційних технологіях. У порівнянні з ієрархічною моделлю, де кожен об'єкт має один попередник і декілька

підлеглих об'єктів, мережева модель дозволяє кожному об'єкту бути одночасно головним та підлеглим об'єктом, тобто взаємодіяти з довільною кількістю інших об'єктів.

Мережеву модель даних також можна представити у вигляді орієнтованого графу, проте відрізняється від звичайного графу наявністю циклів. Це можливо завдяки тому, що вершина може мати декілька батьківських вершин. Така структура даних набагато гнучкіша за порівнянням з ієрархічною моделлю та придатна для моделювання широкого спектру завдань, що можуть виникнути у різних сферах.

Однак, у мережевій моделі доступ до певних даних може тривати досить довгий час, що може бути недоцільним у деяких ситуаціях. Крім того, схеми представлення ієрархічних та мережевих моделей даних можуть бути досить складними, що призводить до складнощів у проектуванні конкретних систем на їх основі.

Складність розробки є однією з головних причин того, що деякі прикладні системи, що ґрунтуються на мережевій моделі даних, можуть перебувати на стадії постійного розроблення або вдосконалення. Незважаючи на це, мережева модель даних є потужним інструментом, що дозволяє розробникам створюв

Реляційна модель є однією з найбільш поширених моделей даних, що використовуються в інформаційних системах. Основним її принципом є представлення даних у вигляді таблиць з рядками і стовпцями, які мають однакові типи даних. Такий формат забезпечує легкий доступ до даних як по рядках, так і по стовпцях, що робить роботу з ними значно простішою і зручнішою.

У реляційній моделі кожна таблиця містить первинний ключ, який є унікальним ідентифікатором для кожного рядка таблиці. Це дозволяє забезпечити точність та надійність збереження даних, а також легко здійснювати пошук та звернення до окремих об'єктів.

Однією з особливостей реляційної моделі є те, що вона дозволяє здійснювати пошук даних за значенням ключових атрибутів, а не за їх структурою, як це

відбувається в ієрархічних і мережевих моделях даних. Це значно спрощує процес пошуку необхідної інформації та підвищує ефективність роботи з даними.

Однак, як і у будь-якої іншої моделі даних, реляційна модель має свої переваги та недоліки. Недоліком є складність побудови відносин між даними, особливо у випадках, коли зв'язки між ними надто складні або взаємозалежні. Також вона може стати надто складною у випадках, коли потрібно обробляти великі обсяги даних.

Об’єктно-реляційна модель (ORM) - це модель даних, яка поєднує переваги реляційної моделі даних і об'єктно-орієнтованої моделі даних. ORM розширює реляційну модель новими типами даних та методами, що дає змогу ефективніше використовувати об'єктно-орієнтовану парадигму для роботи з базою даних.

В ORM дані зберігаються в базі даних і доступ до них здійснюється мовою запитів, так само як у реляційних базах даних. Проте, ORM дозволяє працювати з даними як з об'єктами, що зроблює роботу з ними більш інтуїтивно зрозумілою для програмістів, які використовують об'єктно-орієнтовану парадигму. ORM забезпечує зручний механізм взаємодії між базою даних і програмою, що значно полегшує розробку програмних продуктів.

ORM є потужним інструментом для роботи з базами даних в об'єктно-орієнтованому середовищі і дозволяє ефективно використовувати можливості обох підходів для забезпечення ефективної роботи з даними. Проте, при використанні ORM потрібно враховувати особливості кожної конкретної бази даних та її реляційної моделі, щоб забезпечити оптимальну продуктивність та надійність системи.

Об'єктно-орієнтована модель даних змінює підхід до представлення даних у базі даних. Замість табличного формату, об'єктно-орієнтована модель дозволяє програмістам розглядати сутності бази даних як об'єкти, що зберігаються у оперативній пам'яті. Таким чином, об'єктно-орієнтована модель спрощує розробку

програмного забезпечення, оскільки програмісти можуть працювати з базою даних, використовуючи ті ж мови програмування, що і для інших завдань.

Однією з важливих переваг об'єктно-орієнтованої моделі даних є те, що вона знімає обмеження, які присутні в реляційній моделі. У реляційній моделі кожна таблиця має жорсткий формат, і для зберігання даних потрібно використовувати передбачувані типи даних. У об'єктно-орієнтованій моделі даних немає жодних обмежень на типи даних, що можуть бути збережені. Це дозволяє більш гнучко представляти дані в базі даних.

Ще однією перевагою об'єктно-орієнтованої моделі даних є те, що вона дозволяє програмістам працювати з базою даних в контексті тієї мови програмування, якою вони написані. У реляційній моделі база даних повертає значення у текстовому вигляді, які потім перетворюються у локальні типи даних. У об'єктно-орієнтованій моделі цей етап ліквідований, що спрощує взаємодію програми з базою даних та зменшує кількість помилок.

Для реалізації програмної частини проекту було обрано реляційну СУБД через її більш просту організацію та зручність роботи з даними. Далі буде проведено огляд найбільш популярних реляційних СУБД та визначено СУБД, яка буде використана при розробці програмної частини дипломного проєкту.

PostgreSQL

PostgreSQL [6] - це відкрите програмне забезпечення, яке забезпечує управління базами даних за допомогою об'єктно-реляційної моделі даних. Однією з головних переваг PostgreSQL є те, що вона розширює мову запитів SQL з багатьма корисними функціями, які забезпечують безпечне зберігання та масштабування навіть найскладніших маніпуляцій з даними. Більше того, PostgreSQL відома своєю надійністю, цілісністю даних та потужними можливостями масштабування, які зробили її популярною серед користувачів та розробників програмного забезпечення.

PostgreSQL працює на всіх основних операційних системах, включаючи Linux, macOS та Windows, і має докладну документацію, яка допомагає

користувачам використовувати цю СУБД. Крім того, PostgreSQL володіє потужною спільнотою розробників з відкритим вихідним кодом, яка постійно вдосконалює цю систему та забезпечує користувачам продуктивні та інноваційні рішення.

Загалом, PostgreSQL є однією з найбільш потужних та надійних СУБД на ринку, яка забезпечує широкий спектр функціональних можливостей та допомагає розробникам програмного забезпечення зосередитися на розробці високоякісних додатків.

MySQL

MySQL [7] є однією з найбільш популярних систем управління базами даних в світі, завдяки своїй відкритій архітектурі та багатому набору функцій. Розробка та підтримка MySQL здійснюється компанією Oracle, що дає впевненість у тому, що система постійно оновлюється та розробляється згідно з потребами користувачів.

MySQL є реляційною СУБД, яка підтримує багато операційних систем, включаючи популярні дистрибутиви Linux, MacOS та Windows. Більше того, MySQL забезпечує можливість легкого перенесення даних до інших СУБД завдяки схожості у синтаксисі та архітектурі, що дозволяє розширювати функціональність та ефективність використання системи.

Одною з ключових переваг MySQL є його легкість в вивченні та застосуванні. Багато функцій, доступних у MySQL, зробили його доступним для використання у багатьох популярних мовах програмування. Крім того, у MySQL є велика спільнота користувачів, що забезпечує якісну та оперативну підтримку користувачів, а також допомогу у вирішенні проблем.

MySQL є добре масштабованою СУБД та є практичною у багатьох ситуаціях. Додаткові можливості відстеження та контролю роботи СУБД дозволяють зручно спостерігати за її функціонуванням та забезпечувати високу продуктивність та надійність.

H2 Database

H2 [8] – це реляційна СУБД, яка написана мовою Java. Є досить швидкою, з відкритим вихідним кодом, реалізує необхідні функції та має декілька режимів роботи: вбудований, серверний та змішаний. Необхідно розглянути їх детальніше.

Вбудований режим – у цьому режимі програма відкриває базу даних із тієї самої віртуальної машини за допомогою JDBC(з'єднання з базами даних на Java). Це найшвидший і найпростіший режим підключення. Недоліком є те, що база даних може бути відкрита лише на одній віртуальній машині у будь-який час. Як і у всіх режимах, підтримуються як постійні бази даних, так і вбудовані в пам'ять. У цьому режимі роботи немає обмежень на кількість одночасно відкритих баз даних або на кількість ІАЛЦ.467100.003.ПЗ Арк. 27 Зм. Арк. № докум Підпис Дата відкритих з'єднань, що є безумовною перевагою[7]. Схематичне зображення роботи у вбудованому режимі наведено на рисунку 2.1.

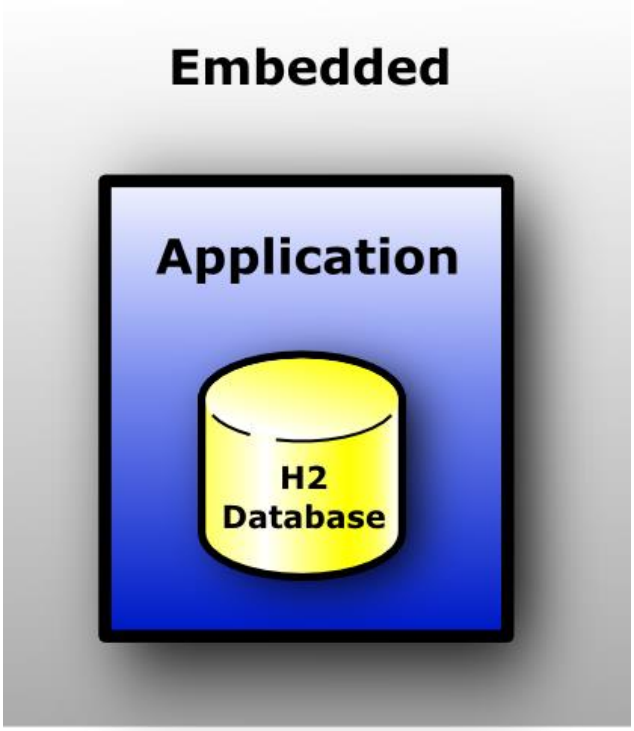

Рисунок 2.1 - Вбудований режим

Серверний режим – При використанні серверного режиму (який іноді називають віддаленим режимом) програма віддалено відкриває базу даних за допомогою JDBC або ODBC(відкрите з’єднання з базами даних). Сервер можна запускати на тій самій або іншій віртуальній машині, або навіть на іншому комп’ютері. Багато програм можуть підключатися до однієї бази даних одночасно, підключившись до цього сервера. Внутрішньо процес сервера відкриває базу даних у вбудованому режимі. Режим сервера повільніший, ніж вбудований, оскільки всі дані передаються через протокол TCP / IP [7]. Також підтримуються як постійні бази даних, так і вбудовані в пам'ять і у цьому режимі також відсутні обмеження на кількість одночасно відкритих баз даних на сервері або на кількість відкритих з'єднань. Схематичне зображення роботи у віддаленому режимі наведено на рисунку 2.2.

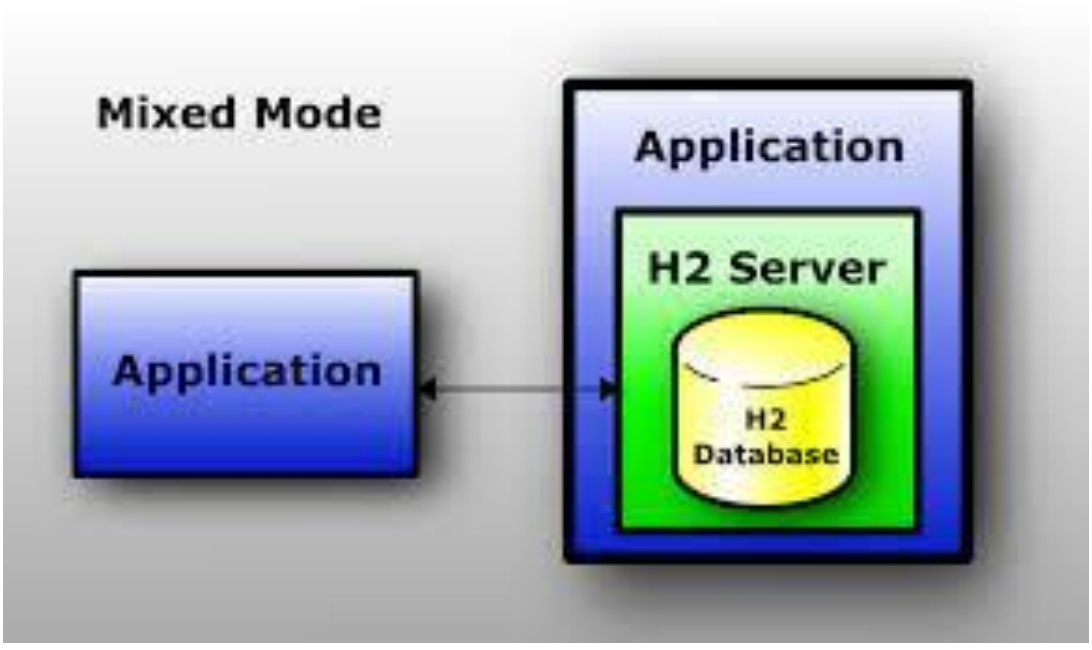

Рисунок 2.2 - Серверний режим

Змішаний режим - це поєднання вбудованого та серверного режиму. Перша програма, яка підключається до бази даних, робить це у вбудованому

режимі, але також і запускає сервер, щоб інші програми (що працюють в різних процесах або віртуальних машинах) могли одночасно отримувати доступ до тих самих даних. Локальні підключення дуже швидкі тому, що база даних використовується лише у вбудованому режимі, тоді як віддалені підключення дещо повільніші [7].

При розробці програмного забезпечення на мові Java можна досить зручно застосовувати цю СУБД тому, що її просто вбудовувати у додатки, що написані мовою Java. Схематичне зображення роботи H2 Database у змішаному режимі наведено на рисунку 2.3.

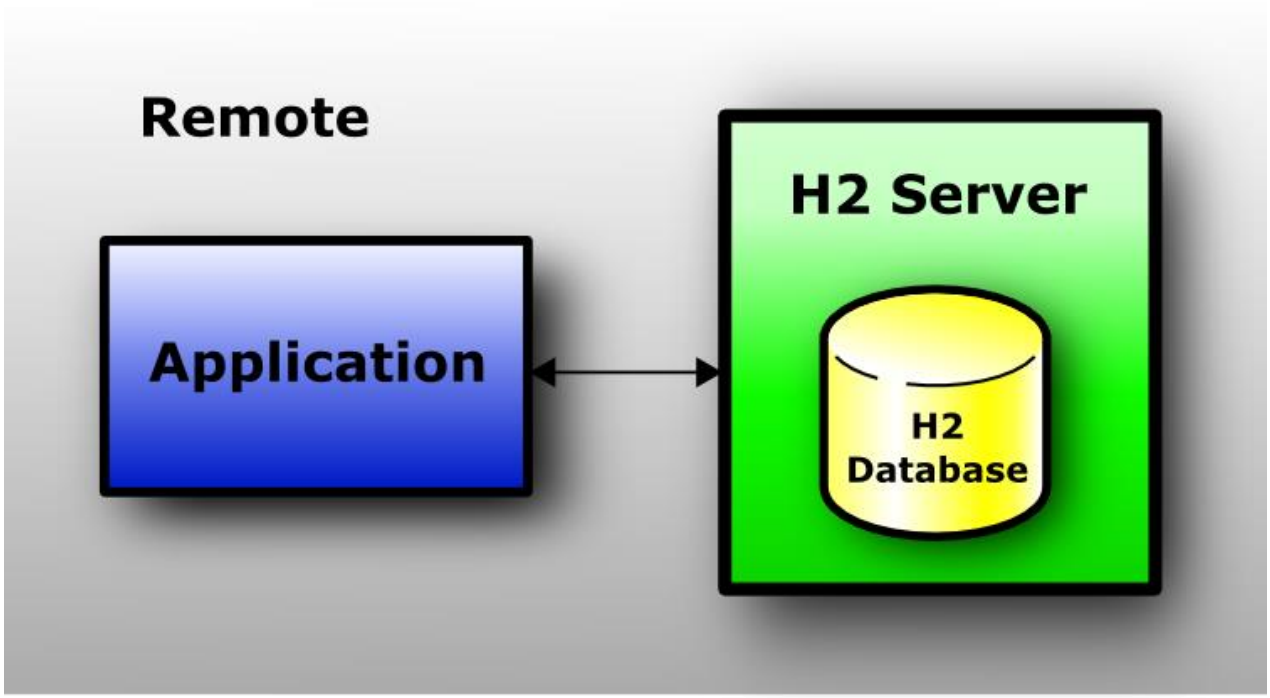

Рисунок 2.3 - Змішаний режим

SQLite

SQLite - це потужна бібліотека, яка реалізує автономний, безсерверний та транзакційний механізм баз даних SQL. Завдяки тому, що код SQLite знаходиться у відкритому доступі, вона безкоштовна для будь-яких цілей.

Однією з особливостей SQLite є те, що вона не використовує парадигму клієнт-сервер, що означає, що рушій SQLite не є окремим процесом, з яким взаємодіє програма користувача. Замість цього, SQLite надає бібліотеку, з якої програма компілюється, і цей рушій стає складовою частиною програми.

Такий підхід дозволяє зменшити накладні витрати, час відповіді і спрощує програму. Більш того, SQLite зберігає всю базу даних в одному стандартному файлі на тому комп'ютері, на якому виконується програма. Простота реалізації досягається завдяки тому, що перед початком виконання транзакції весь файл, що зберігає базу даних, блокується, а ACID-функції досягаються за рахунок створення файлу-журналу.

Кілька процесів або потоків можуть одночасно читати дані з однієї бази, але запис в базу даних можна здійснити тільки в тому випадку, коли жодних інших запитів у цей час не обслуговується. Якщо ж спроба запису відбувається одночасно з іншим запитом, то ця спроба закінчується невдачею, і в програму повертається код помилки. Іншим варіантом розвитку подій є повторення спроб запису автоматично протягом певного заданого інтервалу часу.

В цілому, SQLite - це надійна та ефективна бібліотека для роботи з базами даних, особливо якщо потрібна локальна база даних для програми. Вона має низький рівень відмов, що дозволяє уникнути втрати даних, а також є портативною, тому ви можете використовувати SQLite на різних платформах.

SQLite має простий інтерфейс та широкий функціонал, що дозволяє ефективно виконувати операції з базами даних, такі як створення таблиць, вставка даних, вибірка даних, оновлення і видалення даних, агрегація та ін. Бібліотека має також підтримку транзакцій і запобігає втраті даних у разі виникнення помилки під час операцій з базою даних.

Однак, слід враховувати, що SQLite має свої обмеження і не підходить для великих проектів з великим обсягом даних і високим навантаженням.

Також вона не підтримує деякі функції, які є у великих серверних баз даних, наприклад, розподілені операції і підтримку багатьох користувачів.

Для реалізації проекту до списку технологій було додано таку СУБД, як MySQL. Причинами цього є :

- масштабованість;
- об’єктно-реляційна модель даних, що спрощує розробку;
- потужні можливості для користувача.

2.3. Огляд популярних мов програмування для клієнтської частини

Розробка мобільного додатку - це складний процес, що включає в себе багато різних складових частин. Однією з таких складових є розробка клієнтської частини - фронт-енду. Фронт-енд - це частина додатку, з якою користувачі взаємодіють, тобто інтерфейс користувача.

На сьогоднішній день існує велика кількість популярних технологій для мобайл-розробки, які можуть бути використані для створення фронт-енду мобільного додатку. Основними з них є:

1. React Native [9] є однією з найпопулярніших технологій для розробки мобільних додатків на сьогоднішній день. Вона використовується для створення як iOS, так і Android додатків з використанням JavaScript та React. React Native дозволяє розробляти мобільні додатки з використанням лише одного коду, що робить процес розробки більш ефективним та зменшує витрати на розробку.

React Native забезпечує єдиний інтерфейс користувача для різних мобільних платформ, тому що вона використовує спільний код для iOS та Android додатків. Це означає, що розробникам не потрібно писати окремий код для кожної мобільної платформи, що дозволяє значно зменшити час розробки та складність проекту.

React Native також має велику спільноту розробників та багато ресурсів для навчання, що дозволяє розробникам швидко вивчити технологію та знайти відповіді на свої запитання. Крім того, React Native має багато компонентів, які можна використовувати для швидкої розробки додатків, таких як кнопки, текстові поля, списки та інші.

2. Flutter [10] - це новаторська технологія розробки мобільних додатків, яка створена з використанням мови програмування Dart. Один з головних переваг Flutter полягає в тому, що він дозволяє розробникам створювати красиві та ефективні інтерфейси користувача. Flutter має вбудовану бібліотеку матеріального дизайну, яка надає розробникам готові інструменти та компоненти для створення додатків.

Flutter є швидким фреймворком для розробки мобільних додатків з красивим та ефективним інтерфейсом користувача. Це можливо завдяки його архітектурі та технологіям, які використовуються під час розробки. Схематичне зображення архітектури Flutter наведено на рисунку 2.4

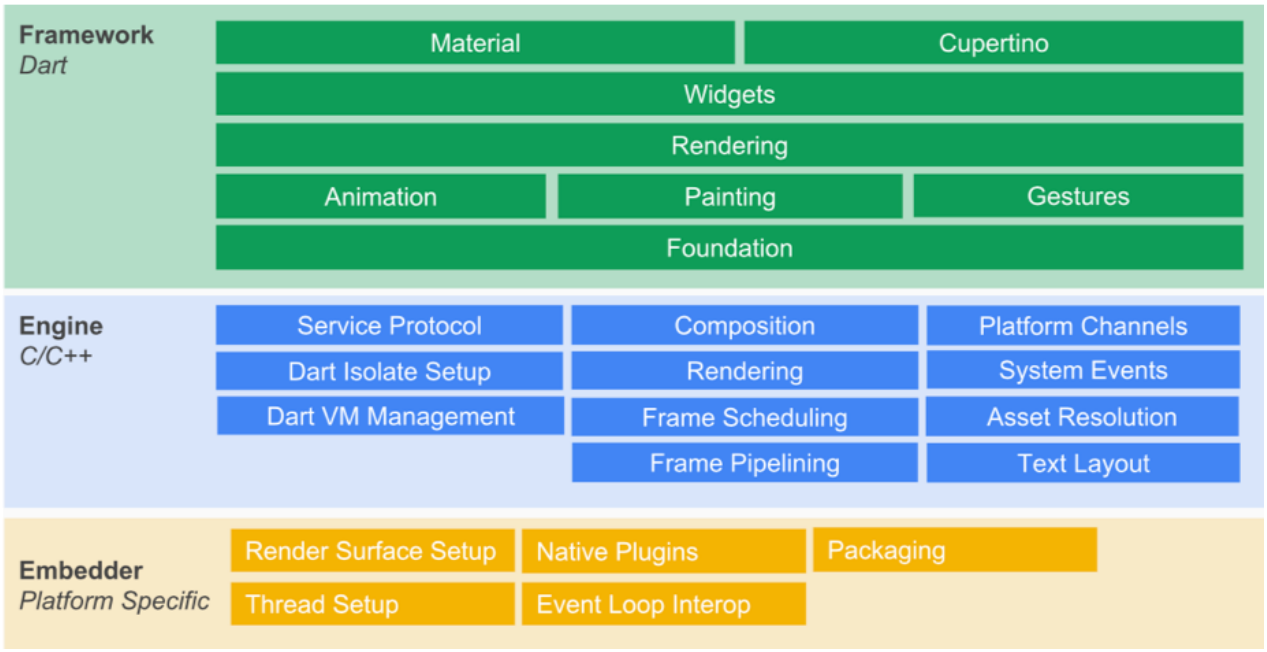

Рисунок 2.4 - архітектура Flutter

Flutter дозволяє створювати додатки для обох платформ - iOS та Android - використовуючи один і той же код. Це забезпечує швидку розробку та зменшення часу на тестування, що дозволяє зосередитися на важливих аспектах розробки, таких як функціональність та дизайн. Крім того, Flutter має велику та активну спільноту, що дозволяє розробникам отримувати швидку допомогу та підтримку в разі потреби.

Зараз Flutter стає все більш популярним серед розробників мобільних додатків, і це не дивно - його можливості та переваги дозволяють створювати високоякісні та ефективні додатки, що задовольнять потреби найвимогливіших користувачів.

3. Ionic [11] є потужним фреймворком для розробки гібридних мобільних додатків. З використанням HTML, CSS та JavaScript, Ionic дозволяє розробникам створювати додатки для iOS та Android з використанням єдиного коду, що зменшує затрати на розробку та забезпечує швидкість розробки.

Ionic використовує веб-технології для створення мобільних додатків, що дозволяє розробникам використовувати знайомий інструментарій для розробки мобільних додатків. Це також означає, що для створення мобільного додатку з Ionic не потрібно володіти спеціальними знаннями, що вимагаються для розробки нативних додатків для мобільних платформ.

Однак, у порівнянні з іншими технологіями, такими як Flutter, Ionic має деякі обмеження. Наприклад, він може працювати повільніше та менш ефективно на старих пристроях, та має обмежену підтримку деяких функцій. Також, гібридна технологія може не завжди забезпечувати таку ж якість роботи додатку, як нативна розробка.

4. Xamarin [12] - це платформа для розробки мобільних додатків з використанням мови програмування C#. Xamarin дозволяє розробникам створювати додатки для iOS, Android та Windows з використанням одного коду, що дозволяє ефективно використовувати час та ресурси на розробку для

кількох платформ. Крім того, Xamarin надає доступ до бібліотек та інструментів Microsoft, які можуть бути корисні для розробки мобільних додатків.

Проте, у порівнянні з Flutter, Xamarin має деякі недоліки, такі як більш важку архітектуру, меншу кількість готових компонентів, більший розмір додатків та меншу швидкість розробки.

Після аналізування різних технологій мобільної розробки, для реалізації проекту було вирішено використовувати Flutter. Це інноваційна технологія, яка дозволяє розробляти ефективні мобільні додатки з використанням мови програмування Dart. Flutter забезпечує швидку розробку та підтримує інтерфейс користувача для різних платформ, що зменшує затрати на розробку та підтримку мобільного додатку в майбутньому.

Отже, використання Flutter є оптимальним вибором для розробки мобільного додатку, який буде ефективним та легким у використанні, а також забезпечить високу якість та швидкість його розробки.

2.3. Огляд допоміжних технологій та фреймворків

Для того, щоб спростити процес розробки серверної частини проекту, основним фреймворком було обрано Spring Framework, який є найпопулярнішим для мови програмування Java та має потужний функціонал, який допомагає програмістам спростити розробку та зробити її більш зручною та зрозумілою. Для розробки програмної частини проекту було прийнято рішення використовувати такі модулі Spring Framework :

- Spring Boot [13] має вбудований сервер Apache Tomcat [14] або Jetty [15], що дозволяє не завантажувати додаткові файли для запуску додатку, зменшуючи тим самим час та зусилля, витрачені на налаштування сервера. Крім того, Spring Boot не потребує XML-конфігурації та дуже простий у налаштуванні та керуванні, завдяки чому програмістам значно легше

працювати з ним. Також він має багато вбудованих функцій, що зменшує кількість написаного коду та збільшує швидкість розробки.

- Spring Data JPA [16] - це один з модулів Spring Framework, який надає розробникам зручні інструменти для роботи з базами даних. За допомогою цього модуля можна значно спростити процес взаємодії з базами даних, а саме створення, зміну і видалення сутностей, виконання різноманітних операцій з даними і багато іншого. Одним із головних переваг Spring Data JPA є можливість записувати об’єкти безпосередньо в базу даних, що дуже зручно та ефективно. Також, за допомогою Spring Data JPA можна виконувати динамічні запити до бази даних, використовуючи опис методів у репозиторії. Це значно спрощує розробку програмного забезпечення і дозволяє розробникам ефективно використовувати свій час. Крім того, використання Spring Data JPA дозволяє досягти покращення читабельності коду та зменшення його кількості. Це досягається завдяки використанню анотацій, які дозволяють автоматично створювати SQL-запити на основі моделей даних, що зменшує кількість коду, який потрібно писати розробникам.

Висновки до розділу 2

Під час проектування та розробки програмної частини було прийнято ряд важливих рішень щодо вибору технологій. Одним з найважливіших виборів було обрання мови програмування, яка б відповідала вимогам проекту та забезпечувала зручність та ефективність розробки. На основі цих вимог було обрано мову Java.

Окрім того, для роботи з базами даних було обрано MySQL - одну з найпопулярніших систем управління базами даних, яка відома своєю надійністю та швидкодією. Для забезпечення зручності та ефективності роботи з базою даних, було обрано фреймворк Hibernate, який дозволяє

автоматизувати процес роботи з базою даних, зменшуючи час та зусилля, необхідні для побудови запитів.

Також для розробки програмної частини були використані певні модулі з Spring Framework - фреймворку, який забезпечує швидку та ефективну розробку різноманітних додатків. Завдяки Spring Framework, можна забезпечити безпеку, контроль доступу та авторизацію, а також налаштувати параметри для доступу до окремих частин додатку.

Для розробки клієнтської частини було вирішено використовувати Flutter та мову програмування Dart. Flutter є однією з найпопулярніших технологій для розробки кросплатформових мобільних додатків. Він має багато корисних інструментів та допомагає забезпечити швидку розробку додатків з високою продуктивністю. Dart є мовою програмування, яка використовується для створення Flutter додатків. Вона має багато корисних функцій та допомагає забезпечити більш високу продуктивність розробки.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі ми розглянемо загальноприйняту архітектуру додатків та наведемо архітектуру програмного додатку, що є кінцевим продуктом дипломного проєкту.

Мобільні додатки зазвичай складаються з двох основних частин: клієнтської та серверної. Клієнтська частина - це той елемент додатку, що відповідає за відображення інтерфейсу користувача. Вона виконується на браузері користувача та забезпечує йому можливість взаємодії з сервером через HTTP запити.

Серверна частина - це та частина веб-додатку, що відповідає за обробку запитів користувача та взаємодію з базою даних. Вона виконується на сервері та забезпечує можливість роботи з базою даних, обробку даних та відповідь на запити користувача.

3.1. Архітектура додатків

Підхід, що досить часто використовується при розробці додатків та має назву «clean architecture» [17] («чиста» архітектура), є одним з підходів до проектування програмного забезпечення, який має на меті відокремити певні частини додатку одна від одної. Такий підхід допомагає створити систему, яка буде легко масштабовуватися за потреби, а також полегшує розуміння структури програмного коду для інших розробників.

Основними ідеями, які лежали в основі створення такої архітектури, були прагнення досягти того, щоб система, яка розроблюється, не залежала від певної технології або фреймворка. Це означає, що фреймворк використовується лише в якості інструмента для розробки системи, а не налаштовується додаток під реалізацію певного фреймворка, враховуючи його особливості та

обмеження. Таким чином, система може бути розгорнута на будь-якій платформі, але фреймворк не буде обмежувати можливості системи.

Ще однією ідеєю, яка стоїть за clean architecture, є збільшення можливостей для тестування додатку. Це досягається тим, що певна частина бізнес-логіки додатку повинна мати можливість бути протестованою без зовнішніх компонентів, таких як база даних або інтерфейс користувача. Це допомагає забезпечити, що система працює правильно і надійно.

Ще одна важлива ідея, яка була використана в clean architecture, - це прагнення до незалежності додатку від користувацького інтерфейсу. За допомогою цього підходу, бізнес-логіка додатку розміщується в самій середині системи, незалежно від того, як користувачі будуть інтерактивно взаємодіяти з додатком. Це дозволяє змінювати інтерфейс користувача без впливу на бізнес-логіку, що допомагає розробникам зберегти робочий код при зміні користувацького інтерфейсу.

Основна ідея чистої архітектури полягає в тому, що програмний код повинен бути організований у логічні шари, які забезпечують відокремлення різних аспектів додатку. Такі шари можуть включати презентаційний шар, доменний шар та інфраструктурний шар. Кожен з цих шарів має свою відповідність та відокремлення від інших шарів, що забезпечує зручність розробки та тестування.

Презентаційний шар включає всі компоненти, які взаємодіють з користувачем, такі як графічний інтерфейс користувача, веб-інтерфейс або інтерфейс командного рядка. Доменний шар включає бізнес-логіку додатку та логіку, яка пов'язана з даними. Інфраструктурний шар відповідає за зв'язок з базою даних, зовнішніми сервісами та іншими залежностями.

Важливим елементом чистої архітектури є так звані «чисті» моделі. Це моделі, які не залежать від зовнішніх залежностей та можуть бути легко тестовані. Чисті моделі зазвичай містять тільки ті властивості та методи, які

необхідні для їх функціонування, і не містять зайвих або незв'язаних компонентів.

Важливо також відокремлювати завдання та використовувати принцип єдиного обов'язку. Кожен компонент повинен виконувати одну та лише одну функцію. Це дозволяє забезпечити простоту та зрозумілість коду, що полегшує його розробку та тестування.

Інший важливий принцип, що використовується в чистій архітектурі, - це інверсія залежностей. За цим принципом, залежності між компонентами повинні бути зворотніми, тобто високорівневі компоненти не повинні залежати від низькорівневих компонентів, а низькорівневі компоненти повинні залежати від абстракцій високого рівня.

На рисунку 3.1 наведено приклад вигляду «чистої» архітектури.

Узагальнюючи, чиста архітектура дозволяє зосередитись на розробці бізнес-логіки додатку, незалежно від інтерфейсу користувача. Це допомагає зберегти робочий код при зміні користувацького інтерфейсу та спрощує розробку та тестування. Крім того, використання чистої архітектури забезпечує зв'язок між компонентами з використанням зворотних залежностей та принципу єдиного обов'язку.

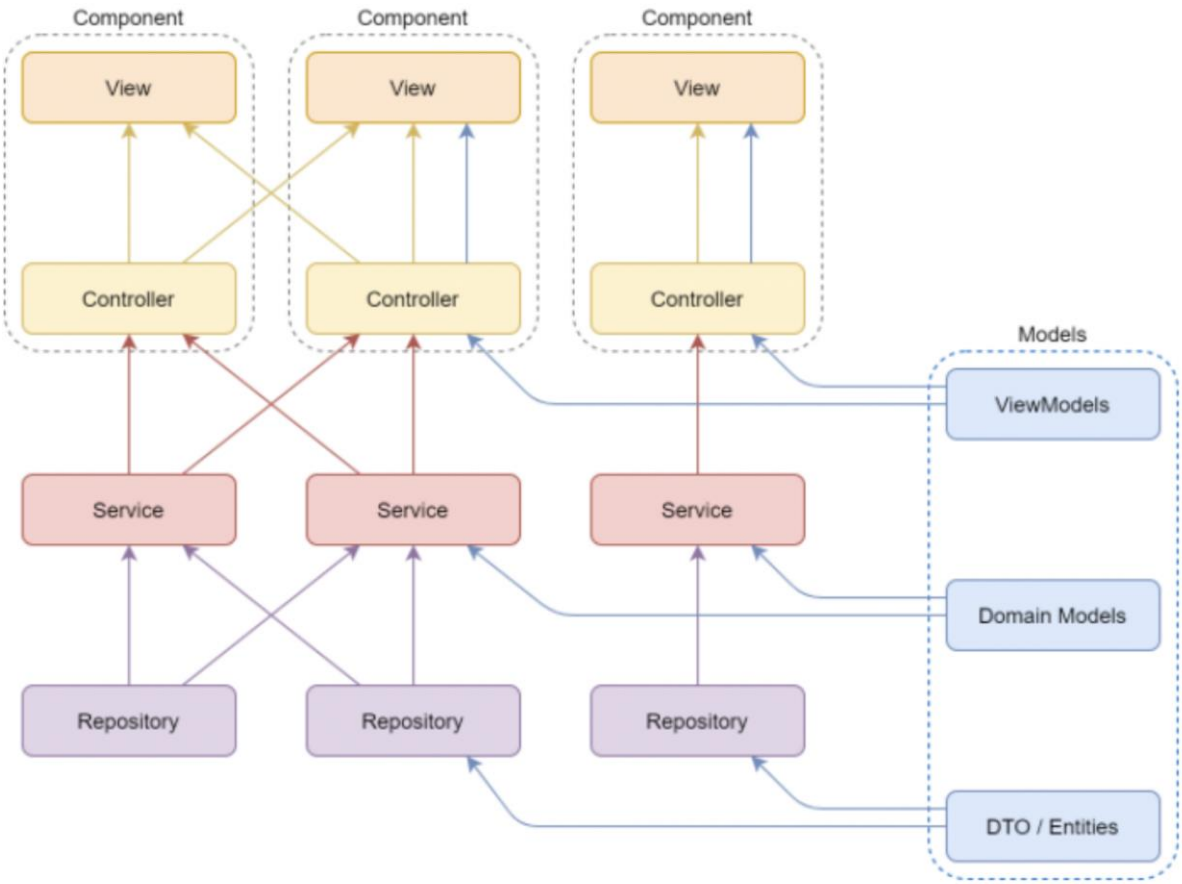

Рисунок 3.1 – Приклад вигляду «чистої» архітектури

3.2. Архітектура серверної частини програмного продукту

При розробці серверної частини програмного продукту було вирішено використовувати Spring Framework та підхід «чистої» архітектури для розробки додатків з урахуванням додаткових елементів архітектури, що сприятимуть розробці якісного програмного забезпечення.

Для покращення читабельності та чіткого розподілу на різні частини додатку було прийнято рішення використовувати окремі директорії для кожного шаблону архітектури. Ці директорії об'єднують файли коду програми, що відповідають цьому шаблону. Таке розподілення файлів дозволяє кожному, хто розглядає код серверної частини, чітко визначити, у якому пакеті знаходиться необхідний файл.

Окрім згаданих вище шаблонів Controller(rest), Service(domain), Repository та Model, було додано також окремі пакети: config, exceptions, mapper та filter. Це рішення було прийнято для того, щоб кожен, хто розглядає код, міг легко знайти потрібний файл і визначити, як саме він впливає на роботу програми.

Пакет config використовується для зберігання конфігураційних файлів, які визначають параметри роботи додатку. В пакеті exceptions знаходяться файли, які відповідають за обробку помилок, що можуть виникнути під час виконання програми. Пакет mapper використовується для зберігання класів, які відповідають за перетворення об'єктів між шаром бізнес-логіки та шаром даних.

Пакет filter містить класи, які відповідають за перехоплення та обробку HTTP-запитів та відповідей. Наприклад, фільтри можуть бути використані для забезпечення безпеки додатку, перевірки прав доступу користувачів або для додавання заголовків до HTTP-відповідей.

Загалом, розподілення файлів за пакетами відповідає принципу «розділення на рівні» (Separation of Concerns) та дозволяє забезпечити модульність та чітку структуру програмного коду. Крім того, використання Spring Framework дозволяє легко інтегрувати різні компоненти додатку та забезпечувати їх взаємодію з використанням інверсії керування (Inversion of Control) та внедрення залежностей (Dependency Injection). Це забезпечує зменшення зв'язків між компонентами, збільшення тестируваності та підвищення швидкодії додатку.

Загальний вигляд архітектури для розробки серверної частини зображено на рисунку 3.2.

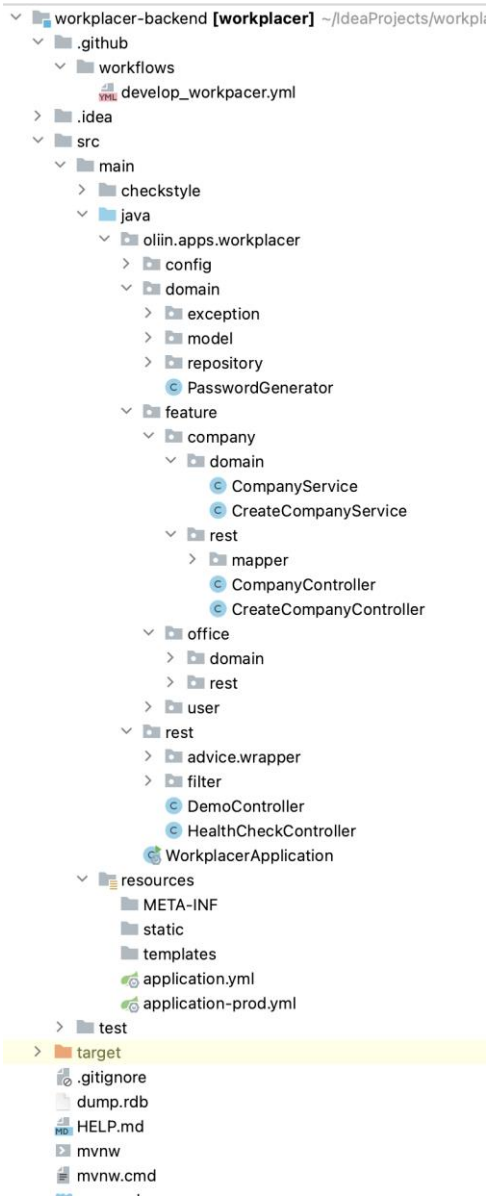

Рисунок 3.2 – Архітектура серверної частини програми

Далі кожен із пакетів буде розглянуто детальніше для пояснення того, що знаходиться у цій частині проєкту.

1. Пакет "rest" у Spring Framework є одним із найбільш важливих компонентів веб-додатку. У цьому пакеті зазвичай знаходяться файли-контролери, які є відповідальними за обробку вхідних запитів від користувачів. Відмінною рисою таких файлів є наявність анотації "@RestController" над

класом, що міститься у файлі. Ця анотація визначає клас обробником вхідних запитів.

Для того, щоб вказати тип запитів, які необхідно обробляти (наприклад, лише ті, які відправлені певним методом, наприклад, POST або GET, чи взагалі всі запити на вказану адресу), у Spring Framework є анотація "@RequestMapping(“/[адреса]”)". Ця анотація дозволяє обробляти всі вхідні запити на вказану адресу. Також існують анотації для кожного методу відправки запитів, такі як "@GetMapping", "@PostMapping" тощо. Після вказання цих анотацій необхідно створити методи, які реагуватимуть на певні дії користувача (наприклад, введення даних, натискання кнопки тощо).

Над кожним методом необхідно вказувати тип запитів, які мають бути оброблені цим методом, а також адресу, за якою необхідно звернутися до цього метода. Головним завданням контролерів є отримання завдання від користувача та передача його до шару бізнес-логіки додатку. Таким чином, контролери є важливими компонентами веб-додатку, які забезпечують обробку вхідних запитів та передачу їх до інших компонентів додатку.

Приклад методу контролера можна побачити на рисунку 3.3. Якщо відповідно використовувати контролери, то можна забезпечити швидку та ефективну обробку вхідних запитів веб-додатку, що є дуже важливим для забезпечення хорошої продуктивності та коректної роботи додатку.

```java
@Slf4j
@RestController
@RequestMapping(path = "/")
@RequiredArgsConstructor
public class HealthCheckController {
    no usages  Artemii Oliinyk *
    @GetMapping
    public ResponseEntity<HealthCheckResponse> checkHealth() {
        log.debug("Health check triggered");

        return ResponseEntity.status(HttpStatus.CREATED).body(new HealthCheckResponse( defaultPassword: "ok"));
    }


    2 usages  Artemii Oliinyk
    public record HealthCheckResponse(@JsonProperty("status") String defaultPassword) {
    }
}
```

Рисунок 3.3 – Метод для перевірки роботи сервера

У цьому прикладі створюється клас HealthCheckController, який є обробником вхідних запитів на адресі "/" тобто на початковій адресі сайту. Метод checkHealth() буде викликаний при відправленні GET запиту на адресу "/", і поверне "status: “ok”". Слід звернути увагу на використання анотації @GetMapping, що вказує на те, що метод checkHealth() повинен бути викликаний тільки при отриманні GET запиту, та на адресу, за якою слід звернутися до цього методу.

У загальному випадку, контролери повинні мати логіку, що виконується при отриманні запиту, та повертати відповідні дані, які зазвичай залежать від даних, що були передані з запитом. Контролери можуть також взаємодіяти з базою даних або іншими компонентами додатку, щоб забезпечити правильну обробку запиту та повернути коректні результати.

2. У пакеті domain містяться файли з класами, які містять методи, що викликаються в контролері. Завдяки цьому, дані з контролера можуть бути передані до класу-сервісу, який не має жодної інформації про клас-контролер. Класи-сервіси мають анотацію @Service, яка дозволяє Spring Framework

знаходити об’єкти інших класів, відмічених анотацією @Service, та виконувати певні зміни за потреби.

У класах-сервісах виконуються різні розрахунки та маніпуляції з даними, які передаються до класів-сховищ (репозиторіїв), що забезпечують роботу із базою даних. Все це дозволяє забезпечити високу роботу веб-додатку, зменшити кількість залежностей між компонентами та спростити їхню розробку та тестування. Приклад методу у класі-сервісі наведено на рисунку 3.4

Рисунок 3.4 – Метод перегляду користувачів компанії

3. У пакеті repository містяться класи, які відповідають за взаємодію з сховищем даних. В даному проєкті цим сховищем є база даних. До цього шару передаються дані із сервісного шару. У Spring Framework репозиторії - це інтерфейси, які наслідують вбудовані інтерфейси Spring Data. Репозиторії необхідно відмітити анотацією @Repository, щоб повідомити контейнер Spring Framework, що цей файл пов'язаний з роботою з базою даних.

Приклад вигляду репозиторія наведено на рисунку 3.5. Цей репозиторій розширює інтерфейс JpaRepository, додаючи до списку методів необхідні специфічні методи для вирішення конкретної задачі. Кожен метод цього репозиторія реалізує операції з базою даних, такі як зчитування, запис або

видалення даних. Таким чином, репозиторії відповідають за збереження даних у базі та їхню обробку відповідно до логіки бізнес-процесу.

```
    Artemii Oliinyk
9   public interface UserRepository extends JpaRepository<User, Integer> {
        1 usage   Artemii Oliinyk
10      Optional<List<User>> findAllByOfficesId(String officeId);
11
        8 usages   Artemii Oliinyk
12      Optional<User> findByEmail(String email);
13  }
```

Рисунок 3.5 – Приклад репозиторія

Кожен репозиторій працює зі своєю сутністю (entity) або моделлю бази даних.

4. Пакет model є одним з основних пакетів у структурі проекту на Spring Framework, і він містить класи-сутності бази даних. Ці класи відповідають за структуру таблиць у базі даних та визначають типи відношень між таблицями. Для створення сутності у базі даних, необхідно відмітити обраний клас анотацією @Entity та @Table(«[назва таблиці]»). Далі, обов'язково потрібно вказати поле, яке буде первинним ключем цієї таблиці, і відмітити його анотацією @Id.

Крім того, для того, щоб включити кожне наступне поле до таблиці, необхідно відмітити його анотацією @Column(«[назва колонки]»). Якщо необхідно пов'язати запис із іншою таблицею, то необхідно над відповідним полем вказати тип зв'язку за допомогою відповідних анотацій, таких як @OneToOne, @OneToMany тощо. Крім того, можна над кожним полем встановлювати різні перевірки, такі як перевірка на порожнє значення, можливість регулювати розмір паролю або перевіряти коректність пошти, за допомогою відповідних анотацій.

Наведений фрагмент коду моделі на рисунку 3.6, який є важливим компонентом створення сутності бази даних у проекті на Spring Framework. Він демонструє приклад використання анотацій для встановлення зв'язків між таблицями та виконання перевірок для полів. Користуючись цими анотаціями, можна встановлювати різні правила для збереження та маніпулювання даними у базі даних.

```
Artemii Oliinyk
@Entity
@Table(name = "companies")
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Company {
    no usages
    @Id
    @GeneratedValue(generator = "uuid")
    @GenericGenerator(name = "uuid", strategy = "uuid2")
    @Column(name = "company_id")
    @Getter
    private String id;
    no usages
    @Column(name = "name")
    @Getter
    private String name;
    no usages
    @Column(name = "active")
    @Getter
    private boolean isActive;
    4 usages
    @ManyToMany(cascade = CascadeType.PERSIST)
    @JoinTable(
            name = "user_companies",
            joinColumns = {@JoinColumn(name = "company_id")},
            inverseJoinColumns = {@JoinColumn(name = "user_id")}
    )
    private Set<User> users = new HashSet<>();

    no usages
    @OneToOne(mappedBy = "company", cascade = CascadeType.PERSIST, fetch = FetchType.EAGER)
    @Setter
    private Office office;
```

Рисунок 3.6 – Фрагмент коду моделі компанії

5. В пакеті exceptions розміщуються класи, що забезпечують користувача конкретною відповіддю при виникненні помилки. Це доцільно, оскільки вимоги до програм можуть включати точні вимоги щодо повідомлень про помилки. Класи, які забезпечують повідомлення про помилки, повинні наслідуватися від класу RuntimeException.

Для того, щоб забезпечити коректне повідомлення, необхідно створити конструктор, який приймає повідомлення про помилку. Далі цей конструктор може викликати інші конструктори відповідно до ієрархії наслідування, яка дозволяє отримати відповідне повідомлення про помилку.

Такі класи можуть бути корисними при роботі зі сторонніми сервісами, наприклад, при невдалому запиті до зовнішньої API або при неможливості підключення до бази даних. Використання таких класів дозволяє програмістам забезпечити коректну обробку помилок та вивести користувачів зі стану незручностей шляхом надання зрозумілих та інформативних повідомлень.

Приклад коду такого класу наведено на рисунку 3.7.

```
5 usages  Artemii Oliinyk
@StandardException
public class UserExistsException extends RuntimeException {
}
```

Рисунок 3.7 – приклад exception класу

6. Пакет config. Цей пакет містить конфігураційні файли, необхідні для налаштування додатку. Основним класом у цьому пакеті є клас SecurityConfiguration, який відповідає за контроль доступу до певних частин додатку.

У класах цього пакету можна створювати об’єкти-біни класів, які необхідні для розробки. Наприклад, можна створити бін для підключення до бази даних або для налаштування зовнішнього сервісу. Такі об'єкти можна створювати за допомогою анотації @Bean та методу, що повертає об'єкт даного класу.

Приклади створення бінів наведено на рисунку 3.8.

```
no usages  Artemii Oliinyk *
@Configuration
@RequiredArgsConstructor
public class ApplicationConfig {

    1 usage
    private final UserDetailsServiceImpl userDetailsService;

    1 usage  Artemii Oliinyk
    @Bean
    public UserDetailsService userDetailsService() { return userDetailsService; }

    no usages  Artemii Oliinyk
    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();
        authenticationProvider.setUserDetailsService(userDetailsService());
        authenticationProvider.setPasswordEncoder(passwordEncoder());
        return authenticationProvider;
    }

    no usages  Artemii Oliinyk
    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration configuration) throws Exception {
        return configuration.getAuthenticationManager();
    }

    1 usage  Artemii Oliinyk
    @Bean
    public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }

}
```

Рисунок 3.8 – Створення об’єктів-бінів

Крім того, у класах цього пакету можна використовувати інші анотації, такі як @Configuration, @EnableWebSecurity, @EnableGlobalMethodSecurity, @Autowired та інші, для налаштування та автоматичної інжекції залежностей.

У розробці даного додатку було використано підхід впровадження залежностей (dependency injection). Це дозволяє спростити код та автоматично додавати необхідні залежності до класів без використання конструкторів або зайвих викликів методів.

Для того, щоб об'єкти необхідних класів були впроваджені у інші за допомогою Spring Framework, клас, об'єкт якого необхідно впровадити, повинен бути відмічений анотацією @Component або її похідними, такими як @Controller, @Service або @Repository. Ці анотації вказують Spring Framework, що клас є компонентом, який може бути впроваджений в інші класи.

Для впровадження залежностей необхідно відмітити залежність анотацією @Autowired. Spring Framework при запуску додатку створить об'єкти класів, відмічених анотаціями @Component або її похідними, та автоматично підставить об'єкт необхідного класу із контексту. Це робить впровадження залежностей дуже зручним і спрощує роботу з об'єктами, що дозволяє більш ефективно розробляти додатки.

3.3. Архітектура клієнтської частини програмного продукту

Архітектура клієнтської частини програмного продукту передбачає використання фреймворка Flutter, який дозволяє розробляти кросплатформні мобільні додатки для Android та iOS. Flutter забезпечує швидку розробку та високу продуктивність додатків завдяки вбудованій системі гарячої заміни коду.

Для ефективного управління станом програмного продукту на клієнтській стороні було обрано патерн State management - BLoC (Business Logic Component). Цей підхід дозволяє зменшити залежності між компонентами програмного продукту та дозволяє краще розподіляти логіку між різними компонентами додатку. Застосування цього підходу дозволяє розділити логіку програмного продукту та відображення на різних рівнях

абстракції. BLoC відповідає за обробку бізнес-логіки, тоді як UI відображає ці дані.

Для роботи з базою даних на клієнтській стороні було обрано Hive - легкий та швидкий, це рішення для локального зберігання даних у мобільних додатках. Hive є надійним та ефективним інструментом для зберігання даних у пам'яті пристрою, забезпечуючи при цьому швидкий доступ до них. Це дозволить зберігати дані у мобільному додатку та забезпечувати швидкий доступ до них навіть у відсутність інтернету.

Додатково до фреймворка Flutter та бази даних Hive було використано різноманітні пакети для реалізації різних функцій та задач в програмному продукті. Серед них можна відзначити такі пакети, як dio для роботи з мережею, easy_localization для локалізації додатку, equatable для порівняння об'єктів, get_it для зручного використання залежностей, logger для виводу логів, rxdart для реалізації реактивного програмування та shared_preferences для зберігання налаштувань додатку на пристрої користувача. Використання цих пакетів дозволяє значно зменшити час розробки та спростити процес створення програмного продукту.

Вміст директорії для розробки клієнту наведено на рисунку 3.9

Для ефективного розвитку клієнтської частини програмного продукту було вибрано структуру проекту по функціям/фічам (feature), яка дозволяє зберігати кожен фрагмент додатку у відповідній папці з файлом сторінки або віджета, кубітом/блоком та стейтом. Ця організація проекту дозволяє швидко знаходити необхідні компоненти, спрощує процес тестування та забезпечує більшу гнучкість у розвитку програмного продукту.

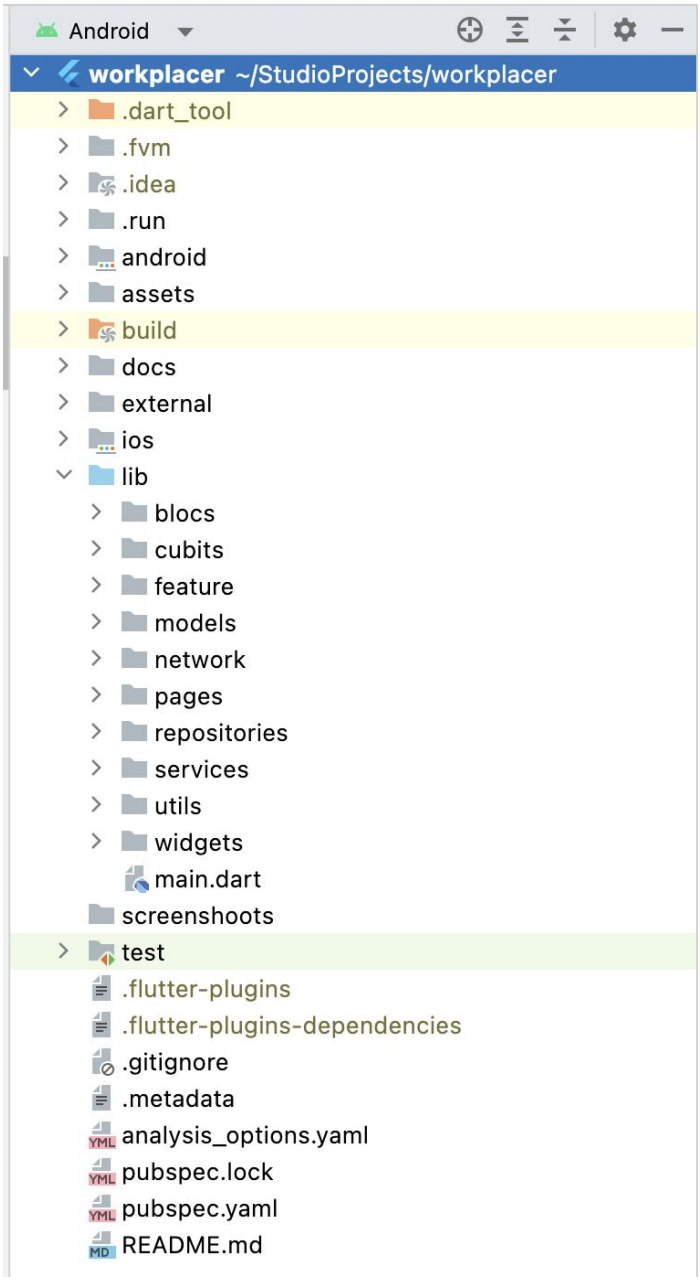

Рисунок 3.9. Структура директорії

Кожна функція або фіча має свою власну папку, яка містить всі компоненти, що стосуються цієї функції, включаючи файли сторінок та віджетів, кубіків/блоків та стейтів. Кожен з цих файлів має свою відповідальність, що спрощує розробку та збільшує масштабованість програмного продукту.

1. Глобальна директорія bloc та cubit відповідають за реалізацію патерну State management - BLoC та Cubit відповідно.

У директорії bloc зазвичай містяться різні блоки, які містять бізнес-логіку програмного продукту. Блоки можуть взаємодіяти між собою та з іншими компонентами програмного продукту за допомогою потоків подій та станів. Крім того, у директорії можуть міститися додаткові файли, які допомагають у реалізації патерну BLoC, наприклад, event, state та repository.

У директорії cubit також зазвичай містяться блоки з бізнес-логікою, проте патерн Cubit дозволяє більш просту та зрозумілу реалізацію стейт-менеджменту за допомогою класів Cubit та State. У директорії можуть міститися файли cubit та state, які містять відповідно логіку та стани програмного продукту.

У обох директоріях також можуть міститися додаткові файли, які допомагають у реалізації State management - наприклад, файли для реалізації інтерфейсу з базою даних, зовнішніх сервісів та інше.

2. Директорія "feature" відповідає за структурування програмного продукту за функціональними модулями, де кожен модуль відповідає за певний функціонал програми. Кожен модуль може містити в собі всі необхідні компоненти для реалізації цієї функції, такі як сторінки, віджети, блоки/кубіти, сервіси та інші.

У директорії "feature" зазвичай містяться піддиректорії з назвою конкретної функції, наприклад, "login", "register", "company", "add_user" тощо. Кожна піддиректорія містить в собі всі компоненти, що відповідають за реалізацію цієї функції.

Структурування програмного продукту за функціональними модулями дозволяє зробити код більш зрозумілим та організованим, сприяє швидкому знаходженню необхідних компонентів та полегшує розробку та тестування окремих функцій програмного продукту.

Вміст директорії “feature” наведено на рисунку 3.10.

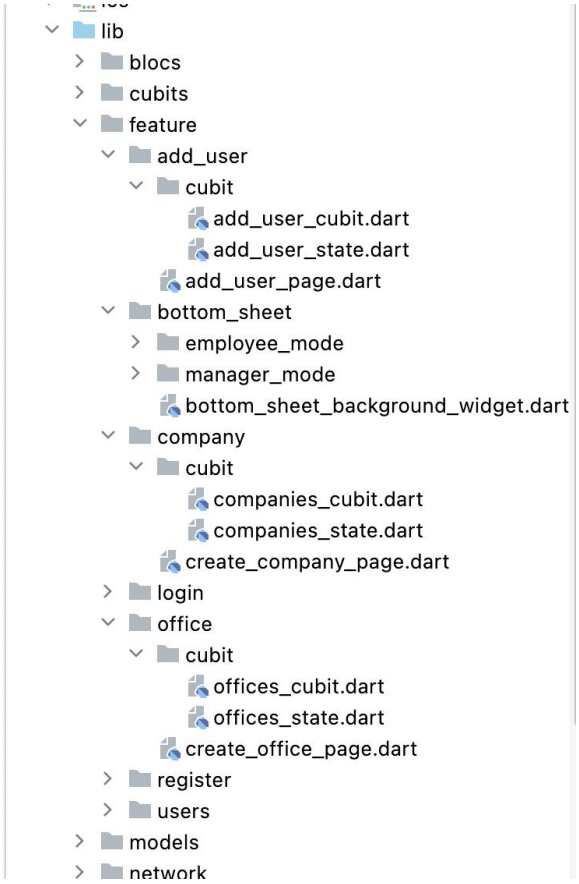

Рисунок 3.10. Структура директорії “feature”

3. Директорія "models" є однією з ключових директорій у проекті на Flutter, оскільки вона містить класи, які описують дані, що використовуються в програмі. Ці класи, також відомі як "data models", використовуються для представлення об'єктів, отриманих з сервера або переданих між різними компонентами програми.

Зазвичай у директорії "models" можна знайти класи для роботи з базою даних, для передачі даних між віджетами, для взаємодії з сервером, а також для обробки даних та їх конвертації. Для полегшення роботи з даними в програмі, можуть використовуватись інші допоміжні класи, такі як класи валідації даних перед їх використанням в програмі.

Вміст директорії “models” наведено на рисунку 3.11.

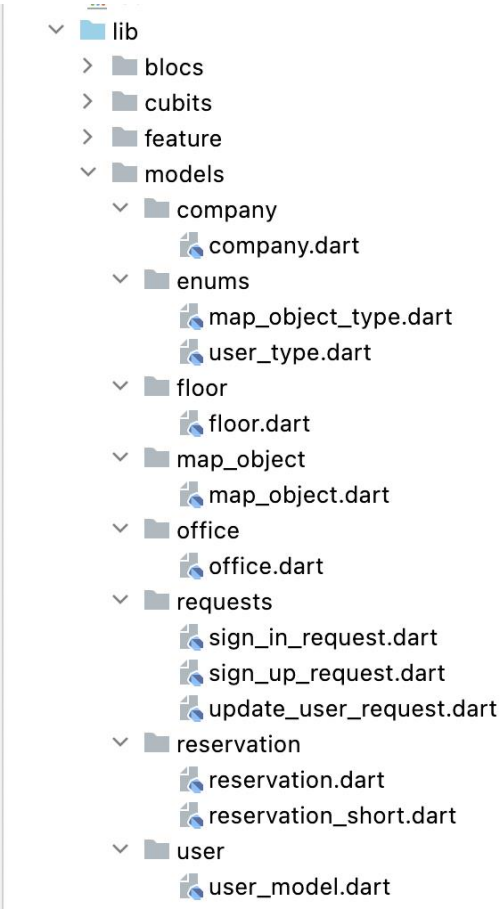

Рисунок 3.11. Структура директорії “models”

Приклад класу моделі наведено на рисунку 3.12.

```dart
import ...

class UserModel extends Equatable {
  const UserModel({
    required this.id,
    this.email = '',
    this.firstName,
    this.lastName,
    this.companies = const [],
    this.offices = const [],
    this.userType = UserType.employee,
  });

  final String id;
  final String email;
  final String? firstName;
  final String? lastName;
  final List<String> companies;
  final List<String> offices;
  final UserType userType;

  @override
  List<Object?> get props =>
      [id, email, firstName, lastName, companies, offices, userType];

  String get username => (firstName == null && lastName == null)
      ? email
      : (firstName ?? '') + ' ' + (lastName ?? '');

  bool get isOfficeManager => userType == UserType.officeManager;

```

Рисунок 3.12. Приклад класу моделі “UserModel”

У кожному класі-моделі присутній метод fromJson(), який використовується для створення нового об'єкта моделі з рядка JSON, та метод toJson(), який перетворює об'єкт моделі на рядок JSON. Це дозволяє зручно передавати дані між клієнтом та сервером в форматі JSON. приклади методів fromJson(), toJson() наведено на рисунках 3.13 і 3.14.

```dart
factory UserModel.fromJson(Map<String, dynamic> json) {
  return UserModel(
    id: json['id'],
    email: json['email'],
    firstName: json['first-name'],
    lastName: json['last-name'],
    companies: List<String>.from(json['company-ids']),
    offices: List<String>.from(json['office-ids']),
    userType: userTypeFromJson(['user-role']),
  );  // UserModel
}
```

Рисунок 3.13. Приклад методу fromJson() в моделі “UserModel”

```dart
Map<String, dynamic> toJson() {
  return {
    'id': id,
    'email': email,
    'first-name': firstName,
    'last-name': lastName,
    'company-ids': companies,
    'office-ids': offices,
    'user-role': userType.toJson,
  };
}
```

Рисунок 3.14. Приклад методу toJson() в моделі “UserModel”

Деякі моделі можуть також мати метод copyWith(), який дозволяє створювати нові об'єкти моделі на основі старого зі зміненими полями. Це допомагає зберегти неінваріантні дані під час модифікації існуючого об'єкту моделі. Приклад такого методу наведено

```dart
UserModel copyWith({
  String? id,
  String? email,
  String? firstName,
  String? lastName,
  List<String>? companies,
  List<String>? offices,
  UserType? userType,
}) {
  return UserModel(
    id: id ?? this.id,
    email: email ?? this.email,
    firstName: firstName ?? this.firstName,
    lastName: lastName ?? this.lastName,
    companies: companies ?? this.companies,
    offices: offices ?? this.offices,
    userType: userType ?? this.userType,
  );
}
```

Рисунок 3.14. Приклад методу copyWith() в моделі “UserModel”

4. У проекті на Flutter директорія "network" зазвичай відповідає за управління мережевими запитами, що виконуються програмою. Ця директорія містить класи та інші файли, які дозволяють здійснювати HTTP-запити до різних джерел даних, таких як сервери API або бази даних.

У цій директорії можуть міститись файли, що виконують наступні завдання:

- Створення та конфігурування клієнтів для HTTP-запитів
- Оголошення класів-моделей для даних, які будуть отримані з сервера
- Реалізація методів для здійснення запитів з використанням різних типів запитів, таких як GET, POST, PUT, DELETE та ін.

- Обробка та обробка помилок, що виникають під час здійснення запитів

Директорія "network" допомагає впорядковувати код, пов'язаний з мережевими запитами та спрощує розробку додатку, дозволяючи розподіляти логіку запитів між різними компонентами програми. Також, у цій директорії можуть бути використані різні бібліотеки для роботи з мережею, наприклад, Dio, Retrofit, HTTP та інші.

Вміст директорії “models” наведено на рисунку 3.15.

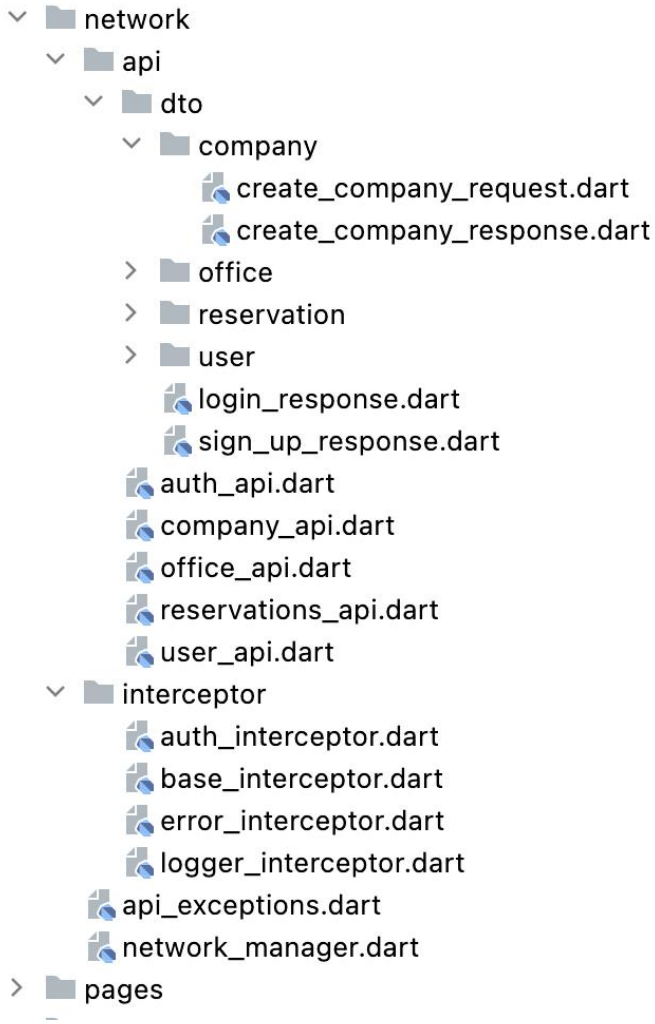

Рисунок 3.15. Структура директорії “network”

Приклад класу моделі для запиту наведено на рисунку 3.16.

```dart
import 'package:equatable/equatable.dart';

class CreateCompanyRequest extends Equatable {
  const CreateCompanyRequest({
    required this.name,
  });

  final String name;

  @override
  List<Object?> get props => [
        name,
      ];

  factory CreateCompanyRequest.fromJson(Map<String, dynamic> json) {
    return CreateCompanyRequest(
      name: json['name'],
    );
  }

  Map<String, dynamic> toJson() {
    return {
      'name': name,
    };
  }
}
```

Рисунок 3.16. Приклад класу запиту для створення компанії

“Api” класи є ключовим компонентом багатьох сучасних програмних продуктів, оскільки вони відповідають за взаємодію додатку з віддаленим сервером за допомогою REST API [18]. У даний час майже кожен додаток потребує взаємодії з сервером, наприклад, для отримання даних з бази даних, збереження даних на сервері, авторизації користувачів та багатьох інших операцій.

На даному проекті структура "Api" класів складається з інтерфейсу, який описує методи для відправки запитів на сервер та отримання відповідей, та двох реалізацій: реальної та фейкової. Реальна реалізація використовується для

взаємодії з сервером, тоді як фейкова реалізація використовується для локальної роботи додатку та для тестування. Приклад інтерфейсу “Api” класу та його реалізацій наведено на рисунках 3.17, 3.18, 3.19.

```dart
import 'package:dio/dio.dart';
import 'package:logger/logger.dart';
import 'package:workplacer/models/company/company.dart';
import 'package:workplacer/network/api/dto/company/create_company_response.dart';
import 'package:workplacer/network/api/dto/company/create_company_request.dart';
import 'package:workplacer/utils/injection/di_utils.dart';

abstract class CompanyApi {
  factory CompanyApi.create(Dio dio) =>
      false ? _CompanyApiFakeImpl() : _CompanyApiImpl(dio);

  Future<List<Company>> getCompany();

  Future<CreateCompanyResponse> createCompany(CreateCompanyRequest request);

  Future<void> deleteCompany(String companyId);
}

```

Рисунок 3.17. Приклад інтерфейсу “Api” класу для роботи з компаніями

```dart
class _CompanyApiFakeImpl implements CompanyApi {
  final _data = <String, Company>{
    '-1': const Company(
      id: '-1',
      name: 'Google',
    ),
    '-2': const Company(
      id: '-2',
      name: 'Apple',
    ),
  };
  int _count = 0;

  @override
  Future<CreateCompanyResponse> createCompany(
      CreateCompanyRequest request) async {
    final id = '${_count++}';
    await Future.delayed(const Duration(seconds: 1), () {
      _data[id] = Company(
        id: id,
        name: request.name,
      );  // Company
    });  // Future.delayed
    return CreateCompanyResponse(companyId: id);
  }

  @override
  Future<void> deleteCompany(String companyId) async {}

  @override
  Future<List<Company>> getCompany() {
    return Future.delayed(
        const Duration(seconds: 1), () => _data.values.toList());  // Future.delayed
  }
}
```

Рисунок 3.18. Приклад фейкового класу “Api” класу для роботи з компаніями

```dart
class _CompanyApiImpl implements CompanyApi {
  _CompanyApiImpl(this.dio);

  final Dio dio;
  final _logger = DIUtils.get<Logger>();

  static const String _companyEndpoint = '/company';

  /// This method is usually called first.
  @override
  Future<List<Company>> getCompany() async {
    final response = await dio.get(_companyEndpoint);
    return Company.listFromJson(response.data);
  }

  @override
  Future<CreateCompanyResponse> createCompany(
      CreateCompanyRequest request) async {
    final response = await dio.post(_companyEndpoint, data: request.toJson());
    return CreateCompanyResponse.fromJson(response.data);
  }

  @override
  Future<void> deleteCompany(String companyId) async {}
}
```

Рисунок 3.19. Приклад фейкового класу “Api” класу для роботи з компаніями

Директорія "interceptor" у структурі директорії "network" відповідає за інтерсептори у програмному продукті. Інтерсептори - це об'єкти, які дозволяють перехоплювати та змінювати запити та відповіді мережевого рівня. Вони дозволяють встановлювати заголовки запитів, авторизовувати запити, додавати параметри до запитів, обробляти помилки відповідей сервера та багато іншого.

Директорія "interceptor" містить класи, які реалізують інтерсептори для взаємодії з REST API. Наприклад, можуть бути створені інтерсептори для

авторизації запитів, додавання заголовків запитів, перехоплення помилок, тощо.

Інтерсептори є важливим елементом побудови надійних та безпечних додатків, оскільки вони дозволяють відлагоджувати та налаштовувати мережеві запити, що допомагає забезпечувати коректну взаємодію з сервером та зменшує кількість помилок в додатку. Приклад класу інтерсептору наведено на рисунку 3.20

```dart
class LoggerInterceptor extends Interceptor {
  LoggerInterceptor({
    required this.logger,
  });

  final Logger logger;

  @override
  void onRequest(
    RequestOptions options,
    RequestInterceptorHandler handler,
  ) {
    logger.d('HTTP=>: [${options.method}] ${options.uri}'
        ' requestId: ${options.headers[AppHttpHeaders.requestId]}');
    handler.next(options);
  }

  @override
  void onResponse(Response response, ResponseInterceptorHandler handler) {
    final options = response.requestOptions;
    logger.d('<=HTTP: [${options.method}] ${options.uri}'
        ' : ${response.statusCode} : ${response.data}'
        ' requestId: ${options.headers[AppHttpHeaders.requestId]}');
    super.onResponse(response, handler);
  }

  @override
  void onError(DioError err, ErrorInterceptorHandler handler) {
    final options = err.requestOptions;
    logger.e('<=HTTP ERROR: [${options.method}] ${options.uri}'
        ' : ${err.response?.statusCode ?? err}'
        ' : ${err.response?.data ?? '[NO-DATA]'}'
        ' requestId: ${options.headers[AppHttpHeaders.requestId]}');
    super.onError(err, handler);
  }
}
```

Рисунок 3.20. Приклад інтерсептор класу “LoggerInterceptor”

5. Директорія "pages" є решткою старої структури проекту, коли додаток був організований за допомогою великої кількості сторінок (pages) або екранів (screens). Цей підхід до організації коду досить складний для розуміння та підтримки, оскільки з часом з'являється велика кількість сторінок, що призводить до збільшення складності коду.

Після переходу на структуру по фічам, директорія "pages" втратила свою актуальність. Замість цього, функціональність додатку розбивається на окремі функціональні блоки (фічі), кожен з яких має свій власний набір файлів і директорій для організації коду.

Нова структура проекту дозволяє зменшити залежності між функціональними блоками, спрощує розуміння та підтримку коду, а також полегшує розробку нових функцій та їх тестування.

6. Директорія “repositories” відіграє важливу роль в багатьох проектах, особливо тих, що працюють з великими обсягами даних. Вона містить "Repository" класи, які є одними з основних компонентів цієї директорії. Ці класи відповідають за управління даними в додатку та забезпечують кешування даних для зменшення кількості запитів до сервера.

"Repository" класи використовуються для роботи з даними в межах додатку. Вони мають реалізації методів, які дозволяють взаємодіяти з іншими компонентами додатку та базами даних. Однією з важливих функцій "Repository" класів є кешування даних, що дозволяє зменшити кількість запитів до сервера та прискорити відображення даних в додатку.

Крім того, директорія repositories може містити інші класи, які допомагають з кешуванням даних, такі як "Cache" та "CacheManager". Ці класи дозволяють зберігати дані в оперативній пам'яті або на диску для подальшого використання. Структура директорії та приклад “Repository” класу наведено на рисунку 3.21.

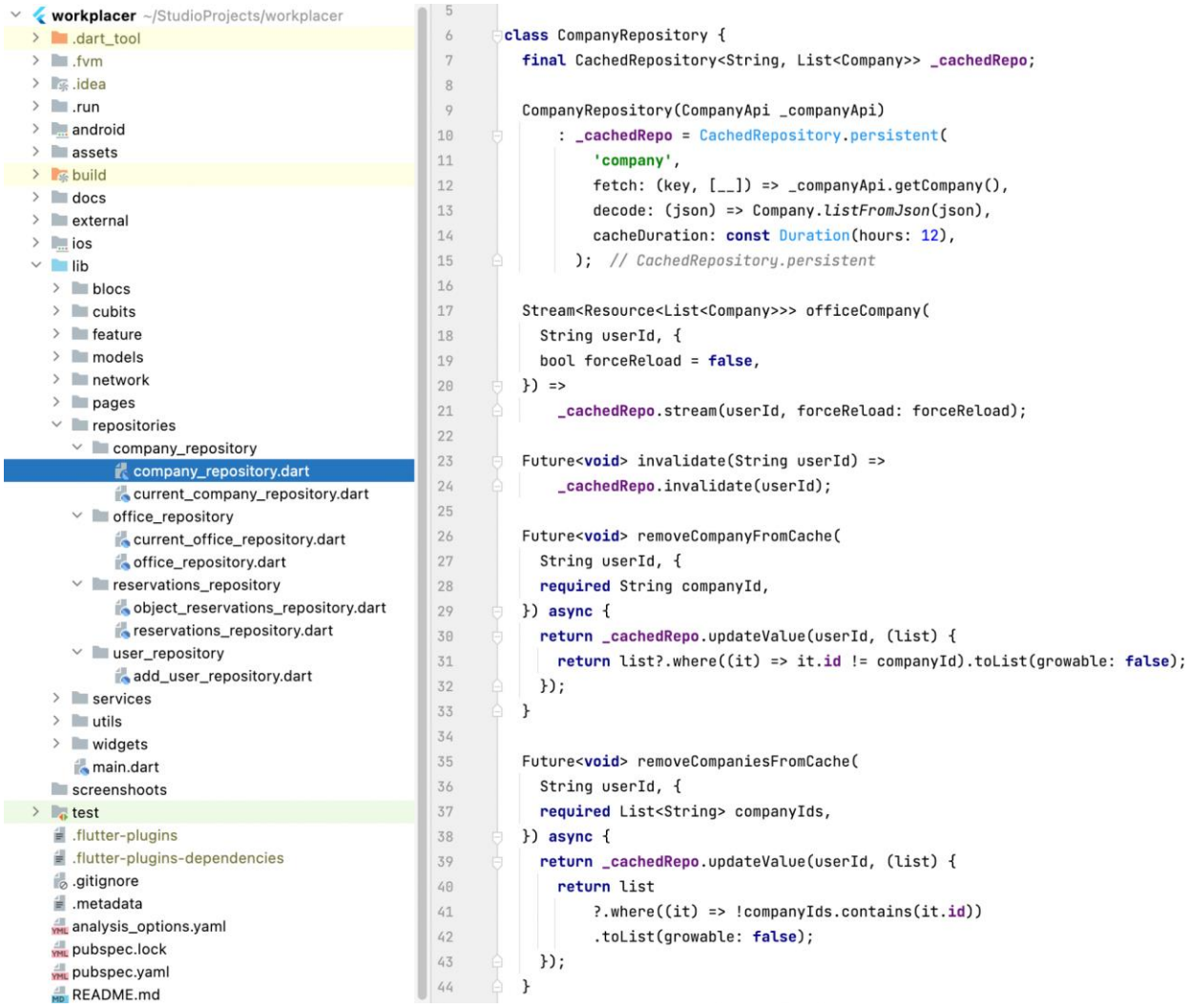

Рисунок 3.21. Структура директорії “repositories” та приклад “Repository” класу “CompanyRepository”

7. Директорія “services” грає важливу роль в багатьох архітектурах додатків. Класи сервісів, які містяться в цій директорії, є прослойкою між різними компонентами додатку. Зокрема, вони допомагають забезпечити взаємодію між Cubit/BLoC класами та Api або Repository класами, в залежності від вимог проекту.

У деяких випадках, коли потрібне кешування даних, класи сервісів можуть використовувати Repository класи. Це дозволяє зменшити кількість запитів до сервера та покращити швидкодію додатку. Крім того, використання

сервісів дозволяє зробити код додатку більш модульним та забезпечити його більшу переносимість та розширюваність. Структура директорії “services” та приклад “Service” класу наведено на рисунку 3.22.

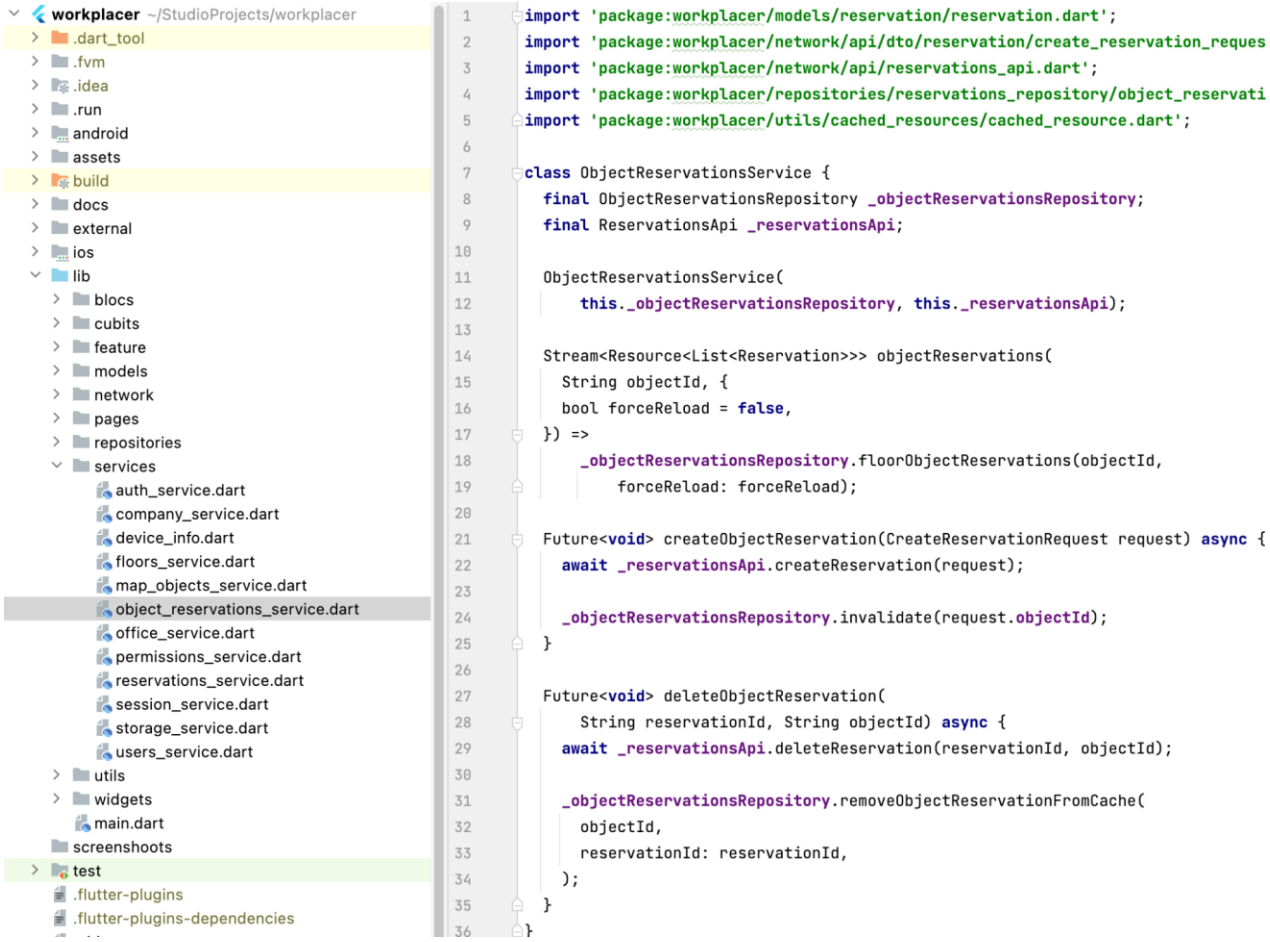

Рисунок 3.22. Структура директорії “services” та приклад “Service” класу “ObjectReservationsService”

8. Директорія "utils" - це дуже важлива частина багатьох проектів, яка містить утиліти та допоміжні класи для спрощення розробки та збільшення ефективності коду. В цій директорії можна знайти різноманітні функції, класи та модулі, що допомагають у вирішенні різноманітних задач.

Зазвичай, в цій директорії знаходяться такі речі, як константи, допоміжні функції та класи, що виконують загальні задачі, реалізації патернів, абстракції, інтерфейси, що дозволяють взаємодіяти з різними модулями та компонентами

проекту. Ці речі зазвичай не пов'язані з конкретним функціоналом додатку, але забезпечують його більш читабельним та підтримуваним кодом. Структура директорії “utils” та приклад допоміжного класу наведено на рисунку 3.23.

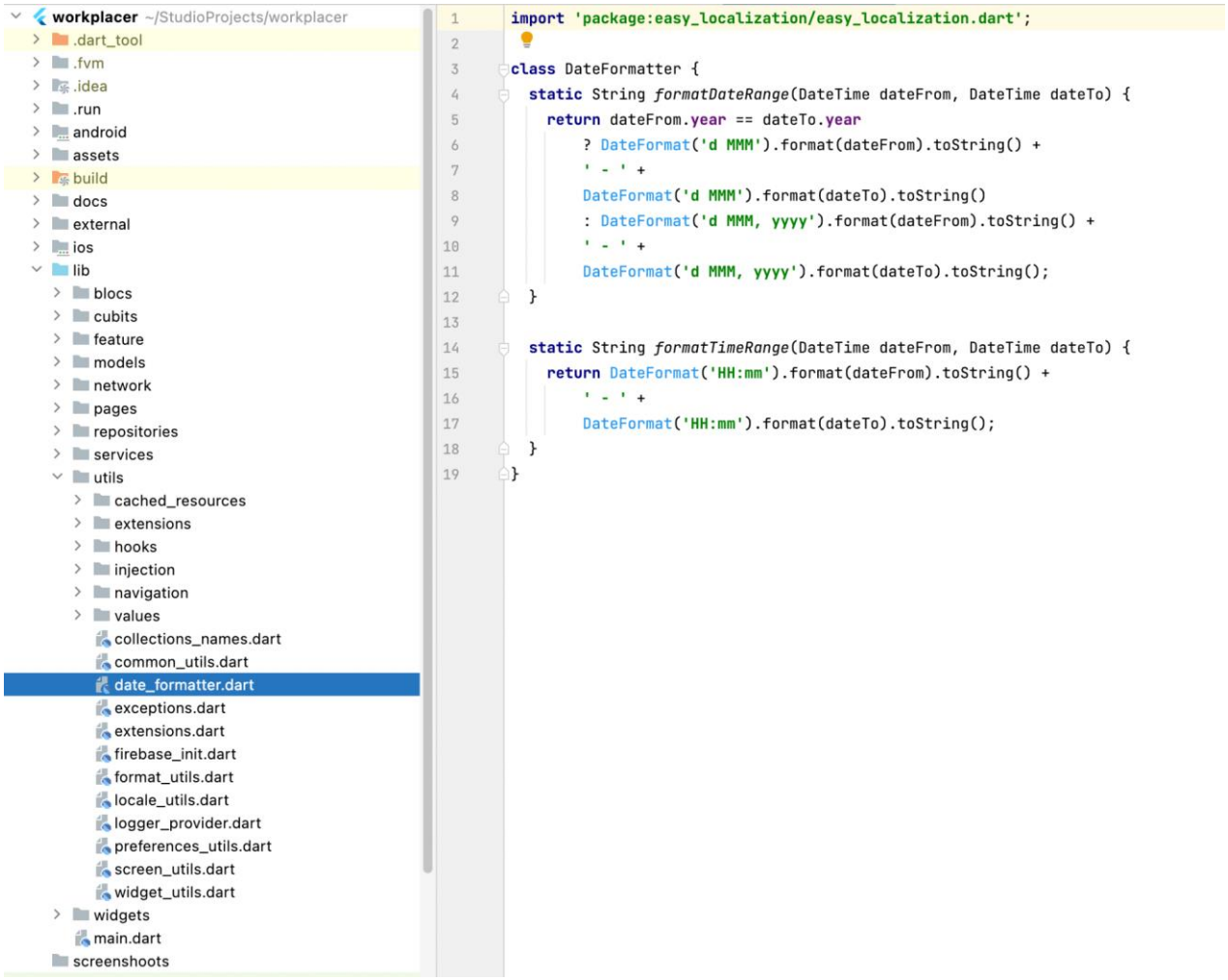

Рисунок 3.23. Структура директорії “utils” та приклад допоміжного класу “DateFormatter”

9. Директорія "widgets" містить багато корисних і загальних віджетів, які можуть бути використані по всьому проекту, зменшуючи кількість повторюваного коду та полегшуючи розробку. Ці віджети можуть містити різні елементи інтерфейсу, такі як кнопки, індикатори завантаження, інформаційні блоки тощо. Крім того, директорія "widget" може містити віджети, які

реалізують спеціальні функції, наприклад, віджет пошуку, віджет відображення дати та часу, віджет графіків та діаграм тощо. Ці віджети дозволяють зосередитись на логіці додатку, не думаючи про деталі відображення елементів інтерфейсу. Структура “widgets” директорії та приклад класу-віджету наведено на рисунку 3.24.

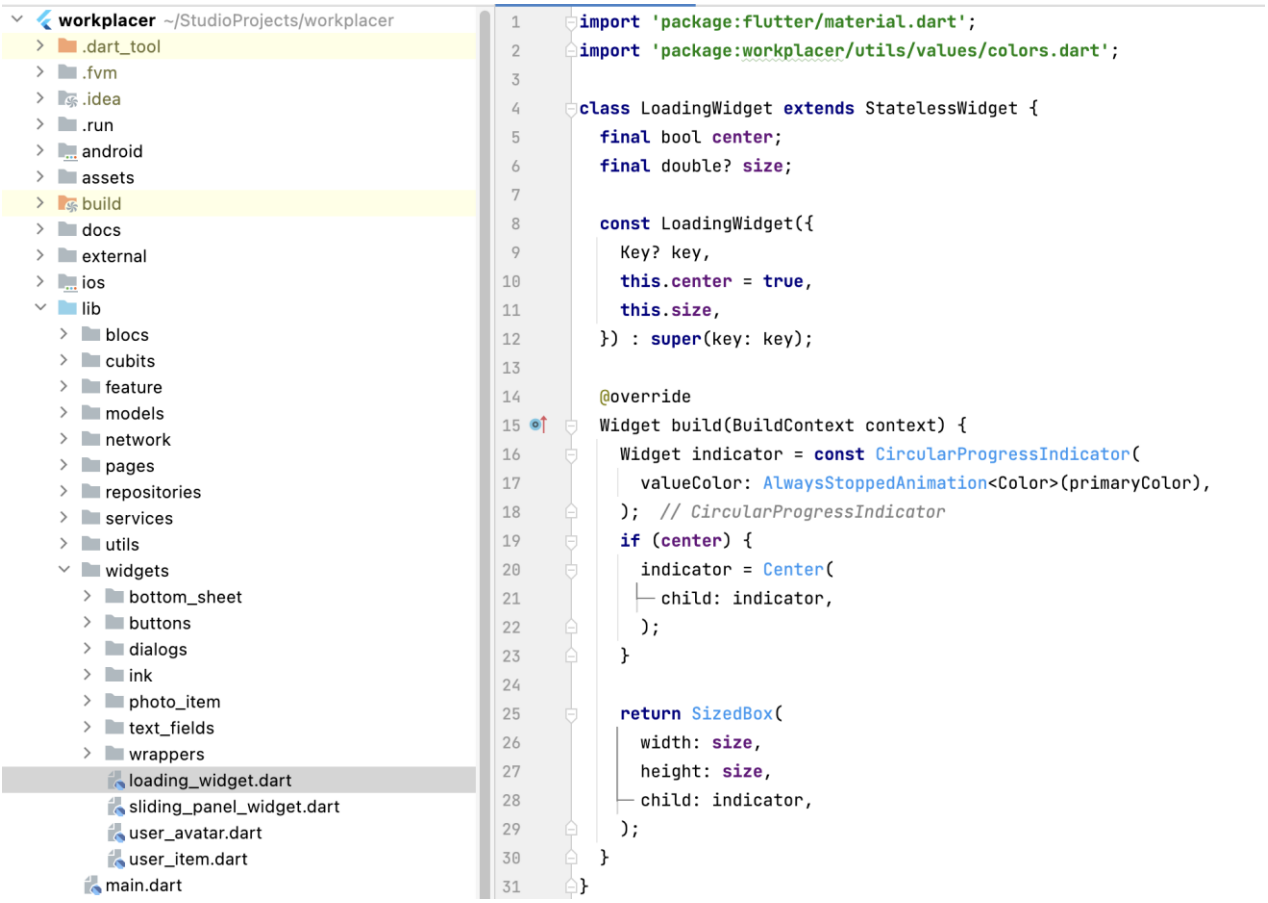

Рисунок 3.23. Структура директорії “widgets” та приклад класу-віджету “LoadingWidget”

Висновки до розділу 3

У цьому розділі ми розглянули архітектуру додатку як на серверному, так і на клієнтському боці. На сервері було використано архітектуру REST API, що забезпечило ефективну та безпечну комунікацію між клієнтом та сервером.

Для впровадження залежностей на сервері було використано Spring Framework та підхід із використанням dependency injection. На клієнтському боці ми використали фреймворк Flutter, який забезпечив розробку кросплатформеного додатку. Ми також використали підхід з використанням пакету Flutter BLoC для управління станом додатку та розбивали додаток на окремі фічі, що сприяло більшій розширюваності та підтримці коду.

Огляд архітектури додатку дозволив нам зрозуміти взаємодію компонентів та покращити ефективність, безпеку та розширюваність додатку. Дані підходи дозволяють швидко та ефективно створювати та підтримувати програмні продукти в майбутньому.

РОЗДІЛ 4
МОДЕЛЮВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

Після успішної розробки програмного продукту було прийнято рішення про проведення моделювання розробленої системи з метою визначення коректності реакції системи на певні дії користувача. Моделювання є важливим етапом у розробці програмного забезпечення, оскільки дозволяє виявити можливі проблеми та недоліки у функціонуванні системи перед її випуском на ринок.

У цьому розділі ми наведемо приклади взаємодії системи з користувачем та проілюструємо функціональність нашої системи, яка є кінцевим продуктом дипломного проєкту. Будуть розглянуті різні сценарії взаємодії з користувачем, що відображають роботу основних функцій системи, таких як авторизація користувача, додавання та видалення столів, створення компанії, офісу, поверху, додавання користувачів і т.д.

4.1. Взаємодія системи з користувачем

Після запуску додатку на мобільному пристрої або комп'ютері, відображається сторінка авторизації, де користувач повинен ввести свої дані для входу в систему. Якщо користувач ще не має облікового запису в системі, він може перейти на сторінку реєстрації, де введе свої персональні дані та створить обліковий запис. Вигляд сторінки авторизації після запуску системи наведено на рисунку 4.1. Сторінка реєстрації наведена на рисунку 4.2.

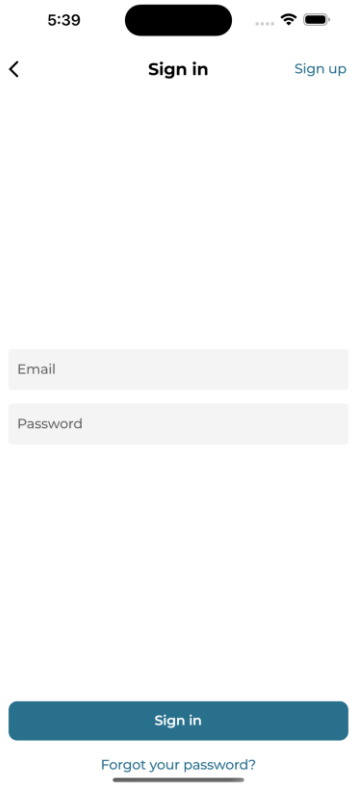

Рисунок 4.1. Сторінка авторизації

Рисунок 4.2. Сторінка реєстрації

Після входу в систему користувач отримує доступ до основної функціональності додатку. Наступна сторінка буде відображена в залежності від того, що потрібно створити для правильної роботи системи:

1. Створення компанії, наведена на рисунку 4.3;
2. Створення офісу, наведена на рисунку 4.4;
3. Створення поверху, рисунок 4.5;
4. Карта поверху. Режим адміністратору без обраного стола продемонстрований на рисунку 4.6, той самий режим з обраним столом на рисунку 4.7. Режим співпрацівника наведений на рисунку 4.8.

Рисунок 4.3. Сторінка створення компанії

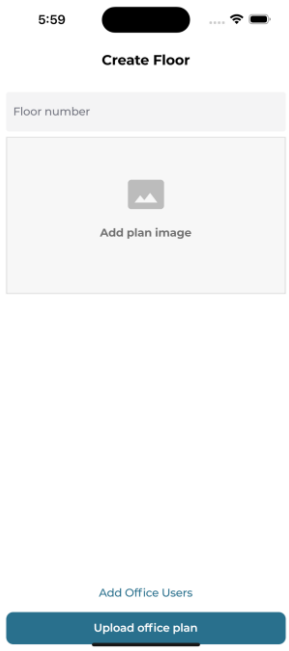

Рисунок 4.5. Сторінка створення поверху

Рисунок 4.4. Сторінка створення офісу

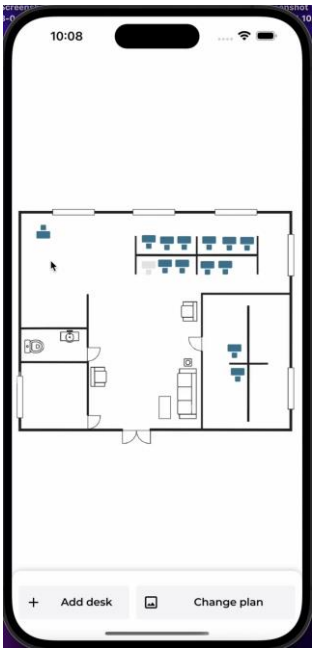

Рисунок 4.6.Карта поверху з режимом адміністратору без обраного стола

Рисунок 4.7.Карта поверху з режимом адміністратору з обраним столом

Рисунок 4.8.Карта поверху з режимом співробітника

Також у додатку адміністратор має можливість додавати нових користувачів на сторінці створення офісу, щоб забезпечити доступ до необхідних ресурсів тим, хто має відповідні повноваження. Це дозволяє ефективно керувати користувачами та їхніми правами, зменшуючи ризик порушення безпеки і підвищуючи ефективність роботи додатку. Інтерфейс сторінок повязаних з додаванням та переглядом користувачів продемострировано на рисунках 4.9. 4.10.

Рисунок 4.9. Сторінка для перегляду користувачів офісу

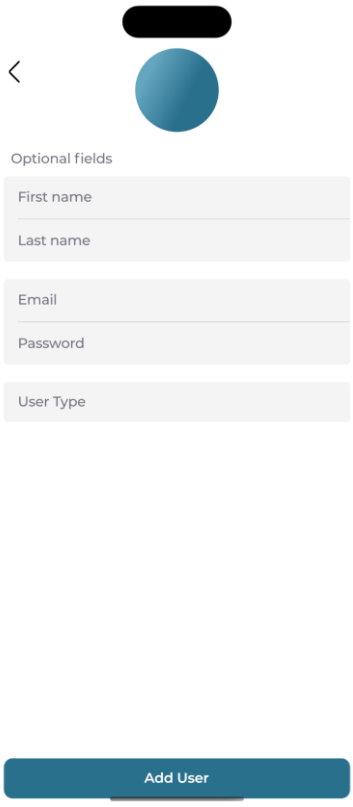

Рисунок 4.10. Сторінка створення нового користувача

Висновки до розділу

Ми успішно реалізували всі вимоги, що були поставлені в завданні. Під час розробки приділили особливу увагу архітектурі нашого додатку, щоб забезпечити його масштабованість та легку розширюваність в майбутньому.

Також забезпечили оптимальну взаємодію між серверною та клієнтською архітектурами, щоб забезпечити ефективний обмін даними та максимальну продуктивність.

Крім того, врахували потенційні проблеми, які можуть виникнути при розширенні функціоналу нашого додатку, і забезпечили належну гнучкість та модульність коду.

ВИСНОВКИ

Розробка програмного продукту потребує досить великих знань та навичок у галузі програмування, а також знання сучасних технологій та фреймворків. У цій роботі було проведено аналіз предметної області, огляд існуючих рішень, підготовку до розробки програмного забезпечення та реалізацію програмного продукту. Розглянуто популярні мови програмування для серверної та клієнтської частини, системи управління базами даних та допоміжні технології та фреймворки. Було також розглянуто архітектуру додатків, серверної та клієнтської частин програмного продукту та взаємодію системи з користувачем.

Результатом роботи є успішно розроблений програмний продукт, який відповідає всім вимогам та потребам. Цей додаток може бути використаний в різних галузях та має потенціал для подальшого розвитку та покращення. Досягнення успішного результату було можливим завдяки використанню сучасних технологій та інструментів, а також ретельному вивченню предметної області та потреб користувачів. Отже, можна стверджувати, що дана робота має великий потенціал для використання в практичній діяльності та є вагомим внеском у галузь програмної інженерії.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Java Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/en/java/
2. Python 3.11.3 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/
3. JavaScript: The Good Parts [Електронний ресурс] – Режим доступу до ресурсу: https://cdn.tc-library.org/Rhizr/Files/daaz74mzphKfnHsen/files/JavaScript-%20The%20Good%20Parts.pdf
4. The Go Programming Language Phrasebook [Електронний ресурс] – Режим доступу до ресурсу : https://the-eye.eu/public/Books/utolica.duckdns.org/David%20Chisnall_The%20Go%20Programming%20Language%20Phrasebook.pdf
5. C Programming Absolute Beginner's Guide [Електронний ресурс] – Режим доступу до ресурсу : https://ptgmedia.pearsoncmg.com/images/9780789751980/samplepages/9780789751980.pdf
6. PostgreSQL 14.7 Documentation [Електронний ресурс] – Режим доступу до ресурсу : https://www.postgresql.org/files/documentation/pdf/14/postgresql-14-US.pdf
7. MySQL Cookbook 3rd Edition [Електронний ресурс] – Режим доступу до ресурсу : http://www.luciopanasci.it/Ebooks/MySQL%20Cookbook,%203rd%20Edition.pdf
8. H2 Database Engine [Електронний ресурс] – Режим доступу до ресурсу : http://www.h2database.com/h2.pdf
9. Learning React Native [Електронний ресурс] – Режим доступу до ресурсу : https://pepa.holla.cz/wp-content/uploads/2016/12/Learning-React-Native.pdf

10. Flutter for Beginners [Електронний ресурс] – Режим доступу до ресурсу : http://livre21.com/LIVREF/F6/F006145.pdf
11. Learning Ionic [Електронний ресурс] – Режим доступу до ресурсу : https://dl.ebooksworld.ir/motoman/Learning_Ionic_2nd.www.EBooksWorld.ir.pdf
12. Xamarin tutorials point [Електронний ресурс] – Режим доступу до ресурсу : https://www.tutorialspoint.com/xamarin/xamarin_tutorial.pdf
13. Spring Boot tutorials point [Електронний ресурс] – Режим доступу до ресурсу : https://www.tutorialspoint.com/spring_boot/spring_boot_tutorial.pdf
14. Apache Tomcat 7 [Електронний ресурс] – Режим доступу до ресурсу : https://www.doc-developpement-durable.org/file/Projets-informatiques/cours-&-manuels-informatiques/Apache/Apache%20Tomcat%207.pdf
15. Learning Jetty [Електронний ресурс] – Режим доступу до ресурсу : https://riptutorial.com/Download/jetty.pdf
16. Spring Data Programming Book [Електронний ресурс] – Режим доступу до ресурсу : https://www.javacodegeeks.com/wp-content/uploads/2016/03/Spring-Data-Programming-Cookbook.pdf
17. CLEAN ARCHITECTURE : A CRAFTSMAN'S GUIDE TO SOFTWARE STRUCTURE AND DESIGN [Електронний ресурс] – Режим доступу до ресурсу : https://yes-pdf.com/book/2542/read
18. REST API Design book [Електронний ресурс] – Режим доступу до ресурсу : https://pepa.holla.cz/wp-content/uploads/2016/01/REST-API-Design-Rulebook.pdf