# РЕФЕРАТ

Пояснювальна записка: ___ с., ___ рис., ___ табл., __ дод., ___ джерел.

Об'єкт розробки: мобільний додаток для допомоги у знаходженні виконавця поставленого завдання

Мета кваліфікаційної роботи: розробити freelance-систему у вигляді мобільного додатку для допомоги у знаходженні виконавця поставленого завдання таких сферах діяльності, як фотографія та відео зйомка, та організації зв’язку між учасниками цього процесу.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної галузі, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано вибір платформи для розробки, виконано проектування і розробка програми, наведено опис алгоритму і структури функціонування програми, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження програми, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення роботи полягає в можливості завдяки розробленому додатку всім користувачам легко та швидко отримувати актуальну інформацію щодо надання послуг спеціалістів з фото та відео зйомки, створювати замовлення, переглядати історії замовлень та редагувати персональні дані у системі.

Актуальність теми кваліфікаційної роботи визначається великим попитом на подібні розробки, через те, що розроблене програмне забезпечення дозволить створити зручну мобільну структуровану систему для вирішення проблеми пошуку спеціалістів для виконання необхідного завдання у таких сферах діяльності, як фотографія та відеозйомка.

Список ключових слів: МОБІЛЬНИЙ ДОДАТОК, ФРИЛАНСЕР, ДОДАТОК, БАЗА ДАНИХ, КОРИСТУВАЧ, ЗАМОВНИК, ВИКОНАВЕЦЬ.

## ABSTRACT

Explanatory note: ___ pp., ___ fig., ___ table, __ appendix, ___ sources.

Object of development: a mobile application to help find the executor of the task

The purpose of the qualification work: to develop a freelance system in the form of a mobile application to help find the executor of the task in areas such as photography and video, and the organization of communication between the participants in this process.

The introduction considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the field of its application, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the analysis of the subject area is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed, requirements to software realization, technologies and software are set.

The second section analyzes existing solutions, selects a platform for development, performs design and development of the program, describes the algorithm and structure of the program, determines the input and output data, provides characteristics of the parameters of hardware, describes the call and download of the program, describes the program .

The economic section determines the complexity of the developed information system, calculates the cost of work to create a program and calculates the time for its creation.

The practical value of the work is the ability to easily and quickly get up-to-date information on the provision of photo and video shooting specialists, create orders, view order histories and edit personal data in the system.

The relevance of the topic of qualification work is determined by the great demand for such developments, due to the fact that the developed software will create a convenient mobile structured system to solve the problem of finding specialists to perform the required tasks in areas such as photography and video.

List of keywords: MOBILE APPLICATION, FREEANCER, APPENDIX, DATABASE, USER, CUSTOMER, CONTRACTOR.

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – бази даних;

ОС – операційна система;

ПЗ – програмне забезпечення;

ПК – персональний комп'ютер;

СКБД – система керування базами даних.

**ЗМІСТ**

# ВСТУП

Сучасний Інтернет простір хоч і має досить структуровану систему, але іноді виникає проблема при пошуку спеціалістів для виконання необхідного завдання у таких сферах діяльності, як фотографія та відеозйомка, тому що не всі задовольняють вимогам клієнта.

Для вирішення даної проблеми було розроблено freelance-систему у вигляді мобільного додатку під назвою PhotoFree. Основною метою цього додатку є допомога у знаходженні виконавця поставленого завдання для клієнтів та організація зв’язку між обома сторонами. Користувач має можливість пройти реєстрацію та авторизацію у системі, створити, або підтвердити замовлення в залежності від сторони, переглянути історії усіх замовлень та редагувати власні персональні дані у системі.

У якості середовища розробки інтерфейсу та структури програми обрано Android Studio версії 3.5.2, що має у своєму складі готові візуальні компоненти для взаємодії користувача з інтерфейсом програми, підтримку мови програмування Java і Kotlin та готовий набір емуляторів мобільних пристроїв з різними версіями операційної системи Android, що призначені для тестування програми та зручності адаптації під різні діагоналі екранів.

Для написання програмного коду обрано об'єктно-орієнтовану мову програмування Java, особливостями якої є простота, близькість до С-подібних мов, а також незалежність від конкретної платформи завдяки компіляції в байт-коді та інтерпретації в Java Virtual Machine. Засоби Java дозволять створити класи та функції для зручної та структурованої організації програмного коду.

Мінімальними системними вимогами до мобільних пристроїв, на яких працюватиме мобільний додаток є операційна система Android, починаючи з версії 8.1 та наявність стабільного Інтернет-зв’язку.

У проектуванні розкриваються деталі і особливості створення додатку з урахуванням обраного середовища розробки, наприклад: макети вікон програми, властивості компонентів, користувацькі класи або файлова

структура. Інтерфейс складається з готового шаблону бічного меню, зображень, додаткових вікон, а також повідомлень. Усі складові інтерфейсу реалізуються за допомогою мови XML, яка має набір основних компонентів та можливість адаптивного відображення. Також проектуванню підлягає база даних, яка розміщується на сервері FireBase та взаємодіє з додатком через Інтернет-зв'язок. Для неї проводиться визначення сутностей, атрибутів, зв'язків і за необхідності виконується нормалізація.

На етапі тестування раніше визначені вимоги зіставляються з результатами реалізації для визначення якості створеного додатку, а також проводиться пошук ймовірних помилок, або непередбачених дій користувача з подальшим їх усуненням.

Програма призначена для спеціалістів та клієнтів у галузі фотографії і відео зйомки.

# РОЗДІЛ 1
# АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

## 1.1. Загальні відомості з предметної галузі

Фрила́нсер або фріла́нсер (англ. freelancer - «вільний копійщик») - вільнонайманець, який сам шукає собі проєкти, може одночасно працювати на декілька фірм. Фрилансер виконує роботу без укладання довгострокового договору з роботодавцем, найманий тільки для виконання певного переліку робіт (позаштатний працівник). Також фрилансером є працівник, запрошений для виконання робіт в ході аутстафінгу. Будучи поза постійним штатом якої-небудь компанії, фрилансер може одночасно виконувати замовлення для різних клієнтів. В Україні фрилансерами переважно називають людей, що виконують будь-яку роботу через інтернет віддалено.

Фрилансер найчастіше сам пропонує свої послуги - через Інтернет, газетні оголошення або користуючись «сарафанне радіо», тобто особистими зв'язками. Фриланс особливо поширений в таких областях діяльності, як журналістика (та інші форми діяльності, пов'язані з написанням текстів), комп'ютерне програмування і дизайн у всіх його проявах (реклама, вебдизайн, дизайн інтер'єру тощо), переклад, різного роду консультаційна діяльність, приватна фото- та відеозйомка.

Ринок фриланс-послуг у даний час вже досить розвинений у Європі і США і стрімко розвивається в Україні та країнах СНД, залучаючи все нових учасників як з боку виконавців, які пропонують свої послуги, так і з боку приватних осіб і організацій, готових до співпраці на віддаленій основі [5].

Плюси фрилансу:

– свобода вибору робочого завдання;

– вільний графік;

– відсутня транспортна проблема - фрилансер працює вдома, в нього немає необхідності кожен день добиратися в певне місце;

– сприятливі умови праці - можна облаштувати робоче місце на свій смак;

– звільнення не загрожує;

– відповідальність лише за свої дії;

– виконання лише власної роботи - ніхто не попросить «підмінити»;

– можливості для міжнародної співпраці.

Мінуси:

– не завжди є гарантії отримання оплати за виконану роботу;

– нестабільність прибутків;

– відсутність соцпакету;

– відсутність оплачуваних вихідних і відпусток;

– необхідність самому вести бухгалтерію і платити податки;

– не завжди є відповідні пропозиції роботи щодо навичок;

– додаткові витрати на робоче місце;

– примарні можливості для якісного кар'єрного росту.

Широку поширеність фриланс отримав з розвитком інтернету: мережа і відповідні інформаційні та банківські технології дозволили деяким категоріям працівників зменшити частоту появи в офісах, або повністю перейти на роботу вдома.

В нинішній час в інтернеті сформувався стійкий прошарок фрилансерів, які заробляють на життя за допомогою віддаленої роботи. Фриланс поширений серед дизайнерів, програмістів, оптимізаторів, копірайтерів, перекладачів, учасників партнерських програм, інженерів-конструкторів. Працює багато спеціалізованих сайтів, покликаних допомогти фрилансерам знайти чергове замовлення.

Схема роботи фрилансера: фрилансери можуть бути прикріплені до одної або кількох організацій-роботодавців, та/або розміщувати своє портфоліо на спеціалізованих сайтах для фрилансерів. Зазвичай, існують як сайти за спеціалізацією у напрямку перекладу, програмування, копірайтингу тощо, так і загальні сайти фрилансерів. З часом, фрилансер може накопичити базу клієнтів

та працювати за індивідуальними замовленнями [17].

Схема дій фрілансера така:

– знаходить замовника;
– виконує завдання;
– отримує гроші.

Дистанційна робота відрізняється від звичної лише тим, що фрилансеру не потрібно їхати в офіс і перебувати в одному приміщенні з роботодавцем.

За деякими оцінками, фрилансери, що працюють віддалено, отримують в 1,5-2 рази більше своїх офісних колег, а часу на роботу при цьому витрачають менше. Крім того, варто врахувати ще час, який витрачається на те, щоб дістатися на роботу і з роботи додому.

При дистанційній роботі у замовника і виконавця немає необхідності зустрічатися один з одним особисто, досить спілкування через Інтернет, телефон, Skype або по електронній пошті.

### 1.2.Призначення розробки та галузь застосування

Основною метою цього додатку є створення freelance-системи у вигляді мобільного додатку під назвою PhotoFree для допомоги у знаходженні виконавця поставленого завдання таких сферах діяльності, як фотографія та відеозйомка, та організації зв’язку між учасниками цього процесу.

Завдяки розробленому додатку всім користувачам надається можливість легко та швидко отримувати актуальну інформацію щодо надання послуг спеціалістів з фото- та відеозйомки, створювати замовлення, переглядати історії замовлень та редагувати персональні дані у системі.

Розроблене програмне забезпечення дозволить створити зручну мобільну структуровану систему для вирішення проблеми пошуку спеціалістів для виконання необхідного завдання у таких сферах діяльності, як фотографія та відеозйомка.

### 1.3.Підстава для розробки

Підставою для розробки кваліфікаційної роботи бакалавра на тему «Розробка мобільного додатку freelance-системи для пошуку виконавця завдання з фото та відеозйомки засобами мови програмування Java в середовищі Android» є наказ по Національному технічному університету «Дніпровська політехніка» від __.__. 2021р. № ____-_.

### 1.4.Постановка завдання

Метою кваліфікаційної роботи є створення freelance-системи у вигляді мобільного додатку для допомоги у знаходженні виконавця поставленого завдання таких сферах діяльності, як фотографія та відеозйомка, та організації зв’язку між учасниками цього процесу.

Для виконання даної мети необхідно спроектувати та розробити мобільний додаток freelance-систему “PhotoFree”, який дозволить автоматизувати пошук можливих виконавців поставленого завдання для клієнтів у галузі фотографії та організувати зв’язок між обома сторонами.

Функціонал даної програми  повинен забезпечувати роботу мобільного додатку: реєстрація, авторизація, робота з власними персональними даними, збереження даних на сервері, меню навігації, робота з замовленнями та перегляд рейтингу користувачів.

Під час проектування даного продукту необхідно створити макети зовнішнього інтерфейсу програми, що складаються з візуальних та невізуальних компонентів з враховуванням того,  що програма призначена для користувачів двох типів, що мають різний набір функціоналу.

Для забезпечення доступу до даних програми необхідно з використанням СКБД  спроектувати базу даних систе6ми.

Обробку даних необхідно виконувати безпосередньо на мобільному пристрої, відображення оновленої інформації повинно забезпечуватися у

реальному часі без необхідності перезавантаження сторінок.

В результаті виконання даного завдання повинно бути отримано файл з розширенням «.apk», за допомогою якого виконується встановлення додатку на пристрій.

Для виявлення та виправлення недоліків та помилок в роботі додатку необхідно виконати тестування його функцій за допомогою емулятора мобільного пристрою, перевірити коректність відображення усіх візуальних компонентів, обробку виключних ситуацій, а також непередбачених дій користувача.

### 1.5.Вимоги до програми або програмного виробу

#### 1.5.1. Вимоги до функціональних характеристик

Програма призначена для спеціалістів та клієнтів у галузі фотографії та дозволить автоматизувати пошук можливих виконавців поставленого завдання та організувати зв’язок між обома сторонами та включає в себе наступні функції:

– реєстрація. При першому відкритті програми кожному користувачеві потрібно пройти реєстрацію, в якій він вказує персональні дані (ім’я, прізвище, номер телефону, місто, зображення профілю), обирає власну сторону (фотограф, або клієнт), та створює дані для авторизації у системі (електронна пошта, пароль);

– авторизація. Щоб авторизуватися у системі користувачеві потрібно ввести електронну пошту та пароль. Якщо дані введені невірно, то з’являється відповідне повідомлення. При повторному відкритті програма автоматично авторизує користувача;

– перегляд власних персональних даних та їх зміна. Після авторизації відкривається вікно програми з персональними даними користувача. Окрім їх перегляду, користувач також може їх змінити, натиснувши на кнопку з відповідною назвою та перейшовши у нове вікно, де у текстових полях

знаходяться ті ж самі дані з можливістю їх редагування. Збереження зміненої інформації потрібно підтвердити за допомогою кнопки. Зміна паролю або пошти відбувається в окремому вікні налаштувань;

– зберігання інформації у базі даних. Уся інформація про користувачів зберігається на сервері, що містить систему авторизації, базу даних, а також хмарне сховище;

– переміщення між вікнами програми. Переміщення відбувається за допомогою бічного меню програми, яке викликається кнопкою у верхньому лівому куті. Верхня частина меню складається з заголовку, в якому міститься ім'я користувача разом з прізвищем, а також електронна пошта. Пункти меню представлені у вигляді кнопок. При натисканні будь-якого пункту, він змінює свій колір, а також відкриває відповідне вікно;

– створення замовлень. Ця функція доступна лише для користувачів, які зареєстровані під стороною клієнтів. При створенні замовлення необхідно ввести дані про дату виконання замовлення, час, тип, тривалість, адресу, оцінку вартості, а також короткий опис, в якому можуть вказуватися побажання клієнта, апаратні вимоги, тощо. Після створення замовлення воно зберігається у базі даних;

– пошук виконавця та прийняття замовлення. Якщо користувач був авторизований у системі як виконавець, то йому буде надана інформація про вільні замовлення, з можливістю їх фільтрації через окреме вікно. Підтвердження відбувається при переході на одне із замовлень та натискання на відповідну кнопку, після чого воно переміщується у розділ прийнятих замовлень. Одне замовлення можуть прийняти декілька виконавців, з яких клієнт в результаті обирає одного з них;

– робота з поточними замовленнями. Обидві сторони можуть по різному взаємодіяти з поточними замовленнями. Сторона клієнтів може переглядати список виконавців, підписаних на дане замовлення, обирати одного для виконання, редагувати замовлення, видаляти, а також змінювати його статус. Статус замовлення може бути двох типів: поточне або виконане.

Статус поточного набувається автоматично. Для сторони виконавців існує можливість відмови від замовлення. В такому випадку воно повертається з вікна прийнятих у вікно вільних замовлень. Для зв'язку між сторонами у вікні перегляду профілю відображається пошта і номер телефону протилежної сторони, а також мається можливість створення текстового чату;

– рейтинг. Після підтвердження замовлення обидві сторони мають можливість одноразово поставити оцінку протилежній стороні у вікні перегляду профілю. Рейтинг кожного користувача складається з п’яти балів, навпроти яких відображається кількість поставлених оцінок, а також середнього значення, яке вираховується з усіх поставлених оцінок;

– налаштування. Останній пункт головного меню призначений для налаштування аккаунту користувача. Вікно містить поля для ведення нової пошти або паролю, а також кнопку для виходу з системи. При її натисканні відкривається вікно авторизації без можливості повернення назад. Зміна даних відбувається лише за умови правильно введеного старого паролю.

### 1.5.2. Вимоги до інформаційної безпеки

Для виявлення та виправлення недоліків та помилок в роботі додатку необхідно виконати тестування його функцій за допомогою емулятора мобільного пристрою, перевірити коректність відображення усіх візуальних компонентів, обробку виключних ситуацій, а також непередбачених дій користувача.

Обробку даних необхідно виконувати безпосередньо на мобільному пристрої, відображення оновленої інформації повинно забезпечуватися у реальному часі без необхідності перезавантаження сторінок.

### 1.5.3. Вимоги до складу та параметрів технічних засобів

Для коректної роботи мобільного додатку визначено мінімальні та рекомендовані вимоги до пристроїв, на яких він працюватиме (табл. 2.1, 2.2). Така конфігурація обґрунтована тим, що приблизно 99% користувачів мають пристрої, на яких встановлена операційна система Android версії 4.0.3 або вище, в результаті чого підвищується доступність додатку.

Таблиця 2.1

**Мінімальні вимоги до пристроїв**

| Назва компоненту | Характеристики |
|---|---|
| Процесор Cortex-A9 | Частота 1 ГГц, 2 ядра |
| Оперативна пам’ять | Об’єм 50 МБ |
| Внутрішня пам’ять | Об’єм 50 МБ |
| Акумулятор | Li-Ion 1000 мА ч |
| Wi-Fi | IEEE 802.11 b/g/n |
| Операційна система Android | Версія 8.0 |
| Екран | 4.19" дюймів, WVGA (480 x 800) |
| Стандарт зв’язку | 2G |

Таблиця 2.2

**Рекомендовані вимоги**

| Назва компоненту | Характеристики |
|---|---|
| Процесор Qualcomm Snapdragon 625 | Частота 2 ГГц, 2 ядра |
| Оперативна пам’ять | Об’єм 100 МБ |
| Внутрішня пам’ять | Об’єм 100 МБ |
| Акумулятор | Li-Ion 3000 мА ч |
| Екран | 5.99" дюймів, WVGA (1920 x 1080) |
| Стандарт зв’язку | 3G, 4G |
| Wi-Fi | IEEE 802.11 b/g/n |
| Операційна система Android | Версія 10.0 |

### 1.5.4. Вимоги до інформаційної та програмної сумісності

Для розробки мобільного додатку застосовано високорівневу об’єктно-орієнтовану мову програмування Java, засоби якої забезпечують зручне та структуроване написання коду. У якості середовища було використано Android Studio, що має підтримку Java і надає можливість редагування зовнішнього інтерфейсу програми через графічний редактор. Збереження даних виконується на сервері FireBase, який має інструменти для реалізації авторизації користувачів, а також набір бібліотек для роботи з Android Studio, за допомогою яких відбувається зчитування, або додавання інформації.

Робота з мобільним додатком передбачає використання бази даних, яка знаходиться на сервері. Тому користувачеві необхідно мати стабільний Інтернет-зв’язок з рекомендованою швидкістю від 1 Мбіт/с, який забезпечується Wi-Fi мережами, або мобільним оператором. При такій конфігурації відправлення та отримання інформації буде здійснюватися своєчасно і без помилок.

# РОЗДІЛ 2
# ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

## 2.1.Функціональне призначення системи

Програма призначена для автоматизації пошук можливих виконавців поставленого завдання з фото та відео зйомки та організації зв'язку між обома сторонами та включає в себе наступні функції:

– реєстрація, авторизація користувачів в системі;
– перегляд власних персональних даних та їх зміна у вікні налаштувань;
– зберігання інформації у базі даних;
– створення замовлень для користувачів, які зареєстровані під роллю клієнтів;
– пошук виконавця та прийняття замовлення;
– робота з поточними замовленнями;
– створення текстового чату;
– формування рейтингу виконавців;
– налаштування аккаунту користувача.

## 2.2.Опис застосованих математичних методів

Однією з основних функцій системи є створення рейтингу фрилансерів, що виконують завдання від клієнтів в даному мобільному додатку. Рейтинг формується наступним чином: після підтвердження замовлення обидві сторони мають можливість одноразово поставити оцінку протилежній стороні у вікні перегляду профілю. Рейтинг кожного користувача складається з п'яти балів, навпроти яких відображається кількість поставлених оцінок, а також середнього значення, яке вираховується з усіх поставлених оцінок.

## 2.3. Опис використаних технологій та мов програмування

Програмний додаток буде працювати на мобільній операційній системі Android версії 4.0.3 та вище. Особливістю цих версій є те, що вони являються універсальною платформою як для мобільних телефонів, так і для планшетів. Під Android працюють велика кількість пристроїв таких компаній як Samsung, Xiaomi, Huawei, Oppo, LG, Lenovo, Google та інших.

До складу операційної системи входить набір бібліотек ядра, які надають більшу частину функціональності бібліотек ядра мови Java.

Платформа використовує віртуальну машину Dalvik. Вона є оптимізованою, регістр-орієнтованою віртуальною машиною. На відміну від неї, стандартна віртуальна машина Java є стек-орієнтованою. Будь-який додаток працює в рамках свого власного процесу і зі своїм власним примірником віртуальної машини. Віртуальна машина Dalvik використовує формат Dalvik Executable (*. Dex). Цей формат оптимізований для використання додатком мінімального обсягу пам'яті. Це відбувається завдяки таким базовим функціям ядра Linux, як організація потокової обробки і низькорівневе управління пам'яттю. Спеціальна утиліта dx, що входить до складу SDK компілює байт-код Java, на якому написані додатки, в dex-формат [19]. Формат установочних файлів має розширення APK(*.apk).

Для роботи з додатками в операційній системі використовується велика кількість бібліотек, таких як:

– Bionic – бібліотека стандартних функцій, несумісна з glibc;

– Media Libraries – мультимедійні бібліотеки на базі PacketVideo OpenCORE (підтримують такі формати, як MPEG-4, H.264, MP3, AAC, AMR, JPEG і PNG);

– SGL – бібліотека двомірної графіки;

– OpenGL ES 1.0 ES 2.0 – бібліотека тривимірної графіки;

– Surface Manager – бібліотека, яка забезпечує для додатків доступ до 2D / 3D;

– WebKit – бібліотека для веб-браузера, може обробляти HTML, JavaScript;

– FreeType – бібліотека обробки шрифтів;

– SQLite – легка СУБД, доступна для всіх додатків;

– SSL – протокол, що забезпечує безпечну передачу даних по мережі.

Розробку додатків для Android можна вести на мові Java (не нижче Java 1.5) з набором інструментів і бібліотек API – Android SDK, призначених для комп'ютерів під операційними системами Windows, Mac OS X і Linux.

Переваги Android:

– відкритість, завдяки чому можна реалізовувати більше корисних функцій;

– установка програм без підключення до інтернету;

– доступність для різних апаратних платформ;

– наявність альтернативних магазинів додатків;

– підтримка багатокористувацького режиму;

– наявність альтернативних прошивок.

Для розробки мобільного додатку було обрано об’єктно орієнтовану мову програмування Java. Вона позиціонується як високорівнева та проста мова, з відсутністю залежності від якоїсь конкретної платформи та однаковою стабільної роботою на процесорах з різною архітектурою.

Цю особливість Java має за рахунок того, що компілятор не генерує машинні команди для процесору, а створює проміжний байт-код для віртуальної машини (Java Virtual Machine). JVM являє собою віртуальний комп’ютер, розміщений в оперативній пам’яті реального пристрою.

Це відрізняє Java від інших мов програмувань, в яких компілятор генерує машинні команди. Це означає, що вихідний код створеної програми не потрібно змінювати або робити перетрансляцію при перенесенні на іншу платформу.

Незалежність від програмного інтерфейсу операційної системи забезпечує набір стандартних бібліотек класів Java, що також називаються пакетами. Вони містять засоби для організації інтерфейсу користувача, роботи з

файлами, мережевими підключеннями та інші. Отже, внутрішня реалізація бібліотек залежить від конкретної платформи і за рахунок цього розробники програмного забезпечення не повинні хвилюватися про різницю в інтерфейсі різних операційних систем.

Але така архітектура Java має й один з головних недоліків, порівняно з мовами програмування, які компілюються у машинний код: продуктивність. Причиною цього є віртуальна машина, яка витрачає значні ресурси комп'ютера, займаючи оперативну пам'ять. І щоб компенсувати цей недолік існує ряд доповнень.

Компілятор JIT (Just-In-Time). Забезпечує компіляцію байт-коду в машинний код в реальному часі при виконанні програми. Його особливістю є те, що при першій зустрічі з фрагментом байт-коду він не тільки передає його до інтерпретатора, а також компілює і зберігає у вигляді машинного коду. І коли програма знову звертається до цього фрагменту, то JIT не викликає інтерпретатор, а забезпечує виконання раніше створеного машинного коду.

Віртуальна машина Java HotSpot. Основною відмінністю від JIT-компіляторів було те, що вона перетворює в машинний код тільки ті фрагменти байт-коду, які впливають на продуктивність. HotSpot проводить оптимізацію коду під час його виконання. Як результат додатки Java працюють з такою ж продуктивністю, як і звичайні програми [8].

Java можна вважати С-подібною мовою, а отже вона має як схожість, так і різницю з мовами програмування Objective C, C++, або C#, тому її опис легше створити при порівнянні з одною с цих мов.

Так як Java – це об'єктно орієнтована мова програмування, то вона в своєму синтаксисі має класи і об'єкти. Однією з особливостей є те, що в класах можуть бути відсутні конструктори, так як компілятор здатний створити автоматично конструктор без параметрів, який в свою чергу викликає конструктор базового класу. Деструктори ж взагалі фактично відсутні, тому що JVM використовує спеціальну програму очищення пам'яті garbage collector (gc) – збирач «сміття». Ця програма запускається періодично і працює у фоновому

режимі. Вона переглядає пам'ять і виявляє невикористовувані об'єкти, які видаляються з пам'яті, як тільки на них не залишається жодного посилання з інших об'єктів.

Також це строго типізована мова, а отже під час компіляції повинен бути відомий тип кожної змінної і кожного виразу. Компілятор перевіряє відповідність типів і запобігає некоректні операції присвоювання. Якщо виникають операції між об’єктами різного типу, при врахуванні того, що в них заздалегідь не визначено перевантаження, то з’явиться відповідне виключення.

Об'єктами в Java є екземпляри класу та масиви. Ім'я будь-якого класу (інтерфейсу може розглядатися як ім'я типу даних. Слід зауважити, що оголошення змінної посилального типу не створює відповідний об’єкт. Для створення об'єкта слід використовувати оператор new, а також значення всіх посилальних типів за замовчуванням дорівнює null.

Стосовно рядків: мова Java не має простого строкового типу. Для роботи з рядками використовується тип(клас) String. Він міститься в пакеті java.lang. Будь-який рядок в Java являє собою об'єкт, до якого застосовні всі дії, що виконуються над об'єктами. Рядки можна об'єднувати, використовуючи оператор конкатенації (єдиний в Java перевантажений оператор «+»). Методи класу, які модифікують об'єкт, насправді не змінюють його, а створюють додатковий об'єкт. В кожному класі визначено метод toString(), який призначений для перетворення об'єктів в рядок (метод успадковується від класу Object).

Java отримала досить немало критики за те, що у ній не реалізовані беззнакові цілочисельні типи даних, які дані часто використовуються в програмах написаних на C. Відсутність цих типів даних може виступити бар’єром при обміні даними між програмами, написаним на Сі, і програмами, написаним на Java. Великі числа без знаку також використовуються в багатьох задачах числової обробки, в  тому числі в криптографії, що робить Java менш придатною для автоматизації подібних завдань. [8].

Ще слід зауважити те, що не можна виконувати приведення цілого типу до типу boolean і навпаки. Значення 0 не є еквівалентом значення false, а ненульова величина – значенню true.

У Java для більшої продуктивності і зрозумілості коду підтримується тільки просте спадкування. Це означає, що будь-який клас-нащадок є похідним тільки від одного безпосереднього базового класу. Така особливість компенсується інтерфейсами. Крім того, в Java допускається кілька рівнів спадкування, що дозволяє створювати ієрархії класів.

Для об'єднання пов'язаних між собою класів, інтерфейсів використовується так поняття, як пакети. Вони забезпечують захищений доступ до класів. В межах пакету дозволений необмежений доступ класів один до одного. Доступ до класу з інших пакетів можна забезпечити за допомогою специфікатору public в оголошенні класу. Також вони створюють власний простір імен (в різних пакетах можна використовувати одні й ті ж імена класів). Кваліфіковане ім'я класу включає в собі ім'я пакета (наприклад, java.applet.Applet). Дана особливість аналогічна бібліотекам C++, проте їх простіше створювати і використовувати. Пакети імпортуються в програму за допомогою оператора import.

При роботі з Java також використовується так поняття, як Java API, що по суті являє собою набір класів, які згруповані в пакети. Версія Java 8 включає набір з восьми пакетів, названих ядром API (Core API):

– java.lang – забезпечує основні функціональні можливості мови (бібліотека вбудованих класів, котрі мають потреби в імпортуванні – Object, String, System, класи-оболонки для простих типів даних і т.д.);

– java.applet – підтримує роботу з апплетами (міні-додатки Java, що працюють під управлінням Web-браузера);

– java.io – стандартна бібліотека вводу-виводу;

– java.util – містить класи, корисні для різних застосувань (загальні утиліти);

– java.net – підтримує мережеві можливості мови Java (роботу в мережі);

– java.awt – забезпечує можливість створення графічного інтерфейсу користувача (Abstract Window Toolkit – абстрактний віконний інструментарій);

– java.awt.image – підтримує обробку зображень;

– java.awt.peer – бібліотека класів, що забезпечують підключення AWT-компонентів в процессеіх створення до реалізацій елементів графічного інтерфейсу, залежать від платформи [8].

Середовищем розробки виступає Android Studio версії 3.5.2. Воно призначене лише для створення мобільних додатків на пристрої, що працюють під операційною системою Android. Така її вузьконаправленість виступає у якості переваги, тому що вона має засоби для реалізації усіх необхідних функцій додатку, а також зручний інтерфейс, який дозволяє швидко адаптуватися.

Вікно середовища розробки складається за таких елементів: головне меню, шлях до активного до активного файлу, дерево файлів проекту, палітра компонентів, дерево компонентів, дизайнер зовнішнього інтерфейсу, вікно властивостей активного компоненту.

В головному меню програми знаходяться усі можливості середовища: від редагування зовнішнього виду програми до налаштування збірки проекту.

Проект в Android Studio складається з файлів Java, що знаходяться у папці з відповідною назвою, та xml файлів, які знаходяться у папці res. Додавати та змінювати файли можна власноруч у дереві файлів проекту.

Дерево компонентів призначене для активізації потрібного компоненту, зміну їх послідовності розташування та внесення або вилучення с контейнерів.

Палітра компонентів призначена для додавання нового компоненту. Це можна зробити двома способами: перетягнути обраний компонент на дизайнер, що знаходиться правіше, або на дерево.

У дизайнері файлу відображений майбутній інтерфейс програми.

В даному редакторі існує можливість переміщувати об’єкти, змінювати їх розмір, встановлювати між ними зв’язки. На панелі інструментів можна відредагувати вигляд самого вікна: обрати необхідну діагональ екран, зміни

орієнтацію або тему додатку, налаштувати переклад, встановити та видалити зв’язки між компонентами.

Вікно властивостей активного компоненту містить у собі більшість властивостей, що доступні для компонента, решту при необхідності можна додати через текстовий редактор xml, в результаті чого вони також почнуть відображатися у вікні. Одне з найголовніших властивостей – це id. Завдяки їй можливо звертатися до компоненту в коді програми, зчитувати та додавати до нього інформацію, або змінювати зовнішній вигляд. Ця властивість визначається автоматично і також може змінюватися самим користувачем, але слід пам’ятати, що при зміні унікального ідентифікатора необхідно відредагувати код програми, де цей компонент використовувався.

В середовищі існує два варіанти для створення інтерфейсу програми: через дизайнер та через текстовий редактор. Переключитися між ними можна у нижній частиині програми. При включенні другого зникнуть палітра, дерево, а також список властивостей компонентів, усе буде відображено у вигляді xml файлу. Головною особливістю текстового редактору є те, що в ньому можливо редагувати властивості, яких не було в дизайнері, а також в ньому знаходяться лише ті властивості і їх значення, які були змінені в процесі роботи, інші ж мають значення за замовчуванням. В такому редакторів користувач працює не з візуальними компонентами безпосередньо, а з мовою XML.

XML – це розширювана мова розмітки, що призначена для зберігання і передачі даних. На перший погляд вона дуже схожа з мовою розмітки HTML, але їх суть та призначення значно відрізняються. XML вирішує завдання зберігання і транспортування даних, HTML же вирішує завдання відображення даних. Таким чином, HTML піклується про відображення інформації, а XML про транспортування інформації. В цій мові існує можливість створення власних тегів. Саме ця властивість визначає головне перевагу, так як це дає можливість адаптувати мову під різні середовища розробки, наприклад в Android Studio існують теги для усіх компонентів та їх властивостей [18].

Перший рядок – це XML декларація. Тут визначається версія XML (1.0) і кодування файлу. Далі описується кореневий елемент документа. В середині нього знаходяться дочірні елементи. Кінець документа визначається закриттям кореневого елемента.

В даному випадку елементами виступають візуальні компоненти додатку. У якості атрибутів тегу виступають властивості, наприклад: колір, текст, розмір, внутрішні та зовнішні відступи.

Під операційною системою Android знаходиться велика кількість пристроїв від різних виробників та з різною діагоналлю екрана. Звичайно при створенні додатку необхідно це враховувати, щоб його інтерфейс відображався коректно не залежно від розміру екрану. Щоб вирішити дану проблему в середовищі Android Studio відсутнє використання звичних пікселів (px), а використовується формат dp (для компонентів) та sp (для тексту на View елементах). dp або dip – Density-independent Pixels – це абстрактна одиниця вимірювання, що дозволяє додаткам виглядати однаково на різних екранах [10]. Окрім цього кожний елемент повинен мати зв’язок з іншими компонентами, або з краями екрану, мінімально повинні бути один вертикальний та один горизонтальний зв’язок (рис. 2.1)

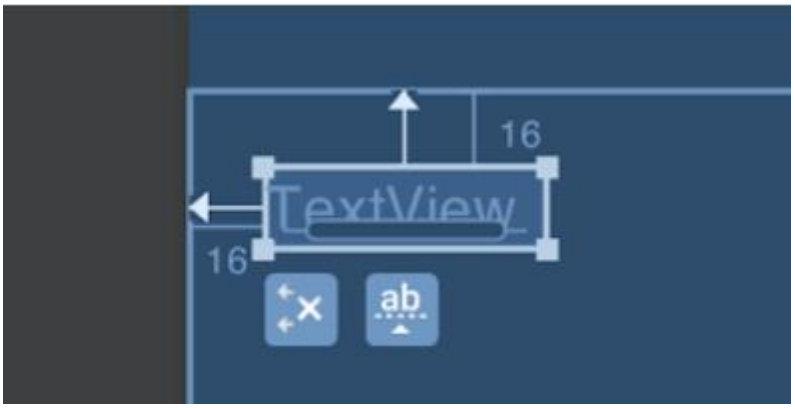


Рис. 2.1.Приклад зв’язку компонента TextView

Це допомагає краще дотримуватися однакових відступів між компонентами під час адаптації. Якщо користувач не створить зв’язки, або не

зробить автоматичне визначення зв’язків, то при запуску додатку на емуляторі, або будь-якому пристрої усі компоненти опиняться у верхньому лівому куті.

Окрім створення зовнішнього інтерфейсу файл XML може зберігати в собі будь-яку інформацію, наприклад значення полів списку (рис. 2.2).

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="typephotos">
        <item>Wedding</item>
        <item>Family</item>
        <item>Fashion</item>
        <item>Kids</item>
        <item>Animals</item>
        <item>Studio</item>
        <item>Business</item>
        <item>Event</item>
        <item>TFP</item>
        <item>Experimental</item>
    </string-array>
</resources>
```

Рис. 2.2. Список значень у файлі XML

В такому випадку зміст файлу представляє собою масив з даними, що відокремлюются тегом <item>. Для перетворення збереженої інформації у масив використовується метод get.Resources().GetStringArray([filename]).

Для відображення списків застосовується компонент ListView. Окрім звичайного відображення масивів інформації у текстовому вигляді зовнішнє представлення елементів може визначати користувач. Для цього використовуються адаптери списків.

Користувацький адаптер – це клас, який розширюється за допомогою класу ArrayAdapter<[тип]> та призначений для перевизначення методів або зовнішнього вигляду звичайного списку.

Основні методи адаптеру:

– конструктор для визначення контексту та значення елементів списку;

– getCount() – повертає кількість елементів у списку;

– getContext() – повертає контекст, в якому знаходиться список;

– getItem(int pos) – повертає елемент списку, що знаходиться на обраній позиції;

– getPosition([тип] item) – повертає позицію елемента;

– getView() – повертає відображення списку (зовнішній вигляд). Кожний з описаних методів може бути перевизначений, але їх назва, параметри, а також повертаємий тип слід залишати вихідними. Адаптер може визначатися за допомогою будь-яких типів: як базових, так і користувацьких.

Зовнішній вигляд елементів адаптеру визначається в окремому файлі XML, що може містити будь-які компоненти відображення. У програмному коді метода getView() можна змінювати дані та властивості компонентів, що надає змогу відображати будь-яку змінну інформацію у кожному елементі списку. Для присвоєння адаптеру списку ListView використовується метод setAdapter().

Присвоїти користувацький вигляд можна будь-яким візуальним компонентами. Для цього необхідно лише створити новй файл XML та присвоїти його як значення для властивості background. Це можна виконати як у програмному коді, так і у візуальному редакторі вікон.

Окрім безпосередньої розробки програмного додатку, середовище Android Studio надає можливість проведення тестування працездатності з допомогою емуляторів (віртуальних мобільних пристроїв). Для їх перегляду необхідно перейти у відповідний менеджер (рис. 2.3).

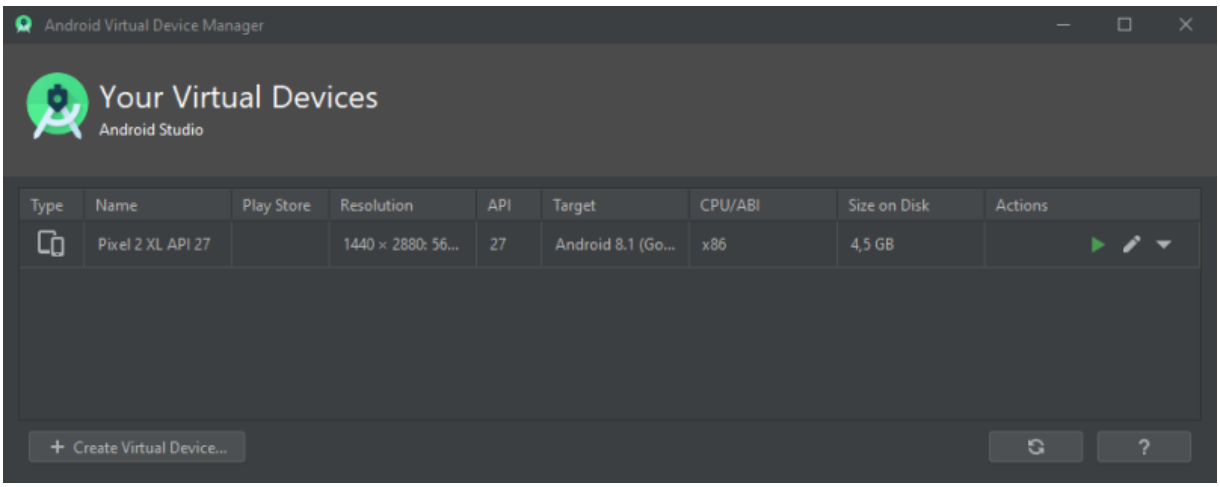


Рис. 2.3. Менеджер емуляторів

Для створення емуляторів використовуються справжні марки та моделі пристроїв, що мають операційну систему Android (Google Pixel, Nexus). У якості прототипів можуть використовуватися не тільки телефони, а й телевізори, розумні годинники, планшети і т.д.

Додання відбувається через конфігурацію віртуальних пристроїв, у вікні якого обирається тип, модель, а також версія операційної системи (рис. 2.4).

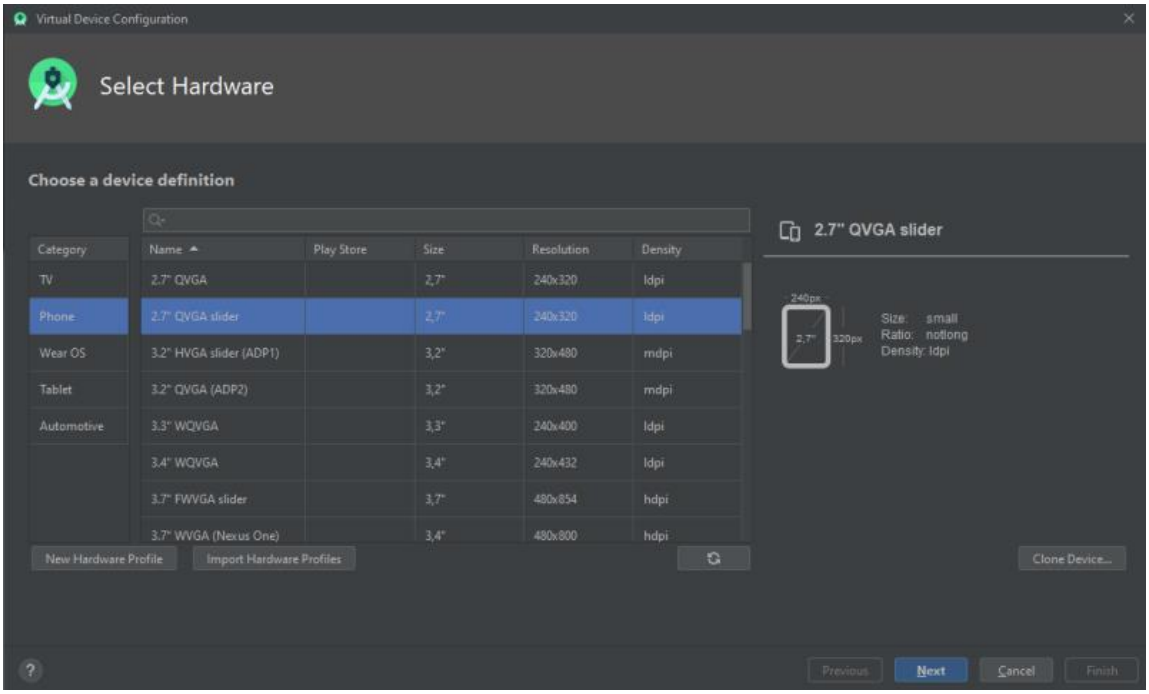


Рис. 2.4. Додання нового емулятору

– Основні характеристики емуляторів:
– тип пристрою;
– назва;
– можливість доступу до сервісів Google Play;
– роздільна здатність екрану (відношення);
– версія API;
– версія операційної системи
– тип архітектури процесора;
– розмір вільної пам’яті на зовнішньому накопичувачі;
– можливі події.

Кожен віртуальний пристрій має набір усіх стандартних функцій телефону, наприклад: доступ до Інтернет-зв’язку, фотографування, введення даних за допомогою екранної клавіатури, відкриття або закриття додатків та інше. Інформація про поточний стан, помилки або трафік Інтернет-зв’язку емулятору відображається у консолі середовища.

Додатково до кожного емулятора закріплене бічне меню (рис. 2.5), в якому присутні можливості включення або виключення пристрою, зміна гучності, створення screenshoot-у екрану, збільшення масштабу інтерфейсу, кнопки «Назад», «Додому», «Диспетчер завдань», а також меню налаштування.

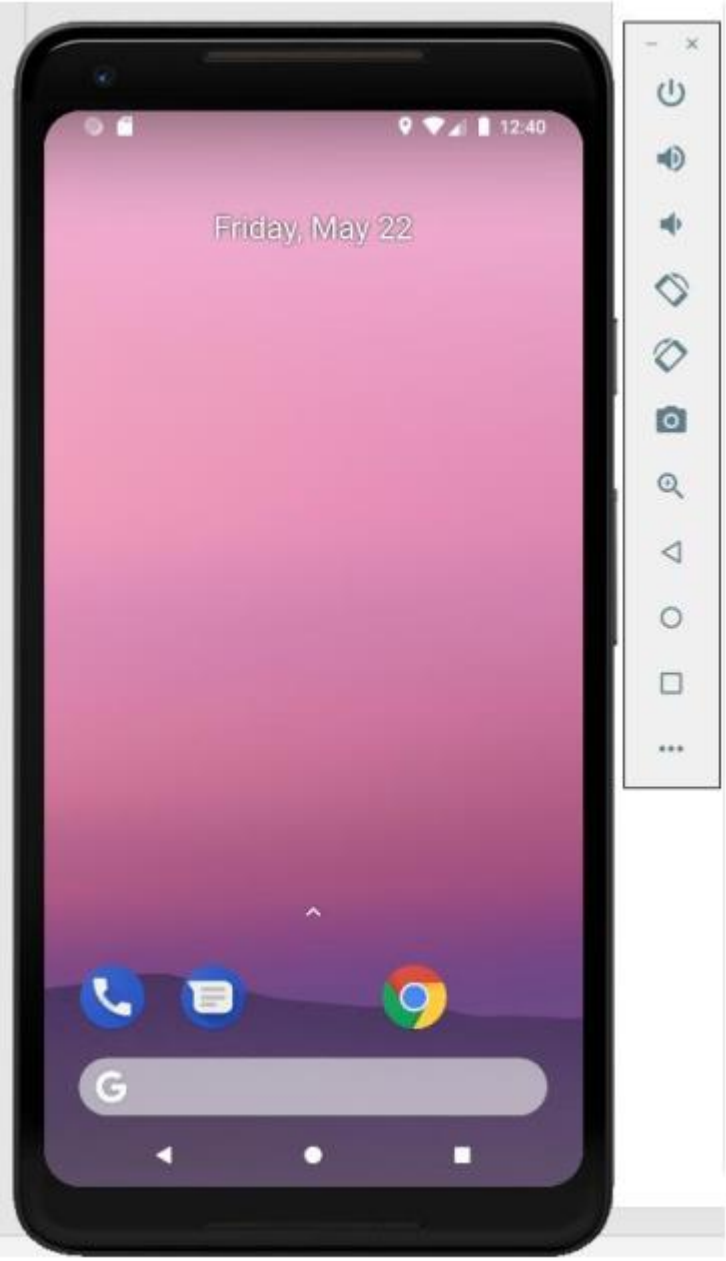


Рис. 2.5. Зовнішній вигляд емулятору

Завдяки гнучкій конфігурації емуляторів існує можливість проведення тестування на різних версіях операційної системи та різних діагоналях екрану.

З недоліків віртуальних пристроїв можна виділити те, що емулятори використовують велику кількість апаратних ресурсів комп’ютера, що можне значно знизити продуктивність, а також при відкритті емулятора для тестування створюється apk-файл, який не придатний для звичайних телефонів, а тому необхідно провести повторну збірку проекту.

Оскільки додаток припускає взаємодію користувачів один з одним, то виникає необхідність зберігання даних на сервері. У якості цього було обрано платформу Firebase. Вона підтримує не лише зберігання даних, а й має вже готовий набір засобів для реалізації проходження авторизації та реєстрації користувача (рис. 2.6).

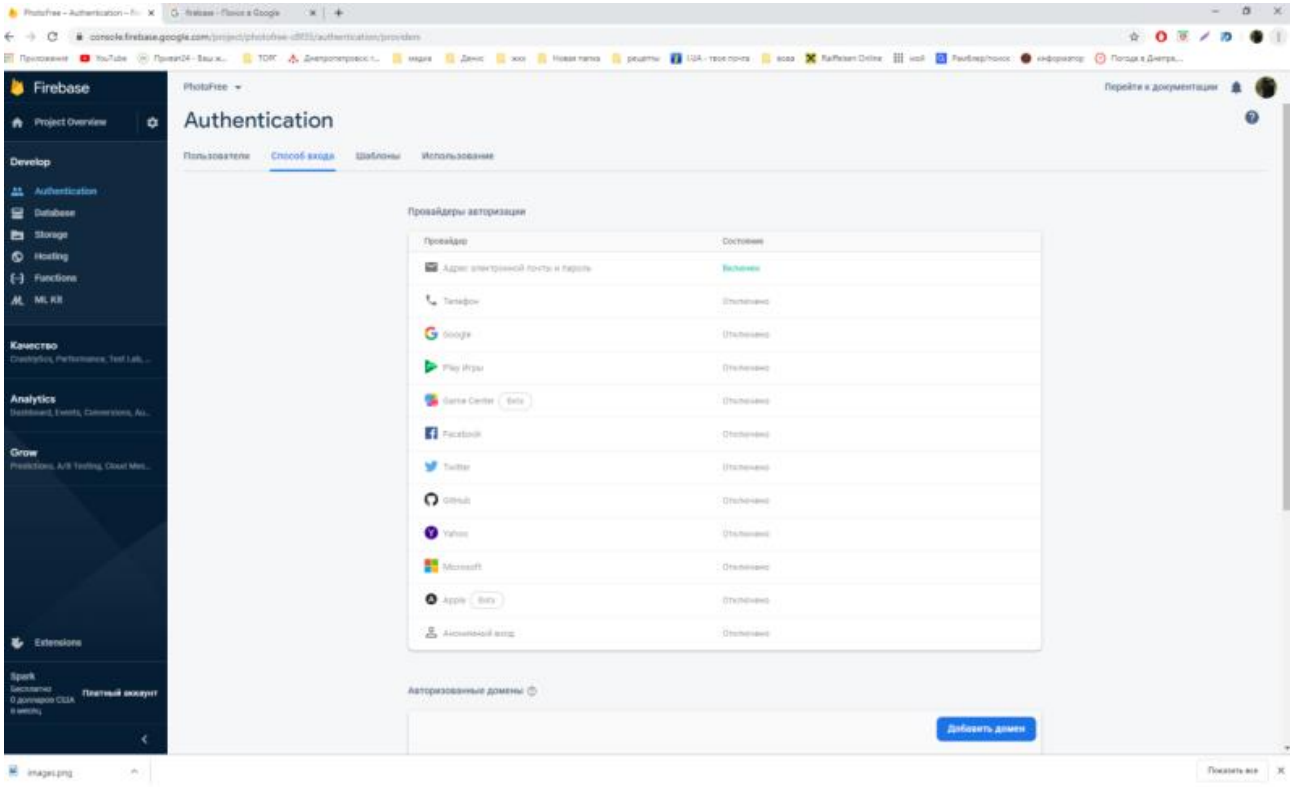


Рис. 2.6. Способи входу для користувачів

Також зберігання інформації в базі даних виконується без застосування SQL (рис. 2.7). Це полегшує роботу, якщо інформація має досить легку структуру, але коли виникає необхідність складної вибірки даних, наприклад з додатковими операторами І, АБО, то потрібно буде створити додаткове поле у кожному записі, що буде відповідати цій умові.

Щоб підключити базу даних спочатку необхідно створити новий проект на сайті Firebase, включити можливість авторизації користувачів за допомогою пошти і паролю, додати базу, та додати власний проект через Android Stuido.

Для підключення проекту необхідно з головного меню програми увійти до вкладки Tools-Firebase. Після цього авторизуватися у системі Google та обрати раніше створений проект. Щоб мати можливість користуватися авторизацією і базою даних потрібно додати сам плагін. Це можна зробити автоматично у діалоговому вікні, натиснувши кнопки “Add Authentication” та “Add Realtime Database”.

Для реєстрації та авторизації у додатку необхідно створити змінну типу FirebaseAuth, в якій існують дві відповідні функції з параметрами email та password [9].

Приклад коду авторизації:

```
mAuth.signInWithEmailAndPassword(email, password)
.addOnCompleteListener(this, new OnCompleteListener<AuthResult>()
{
@Override
public void onComplete(@NonNull Task<AuthResult> task)
{ if (task.isSuccessful())
{
Toast.makeText(MainActivity.this,
"Authentication completed.",
Toast.LENGTH_SHORT).show();
Intent intent = new Intent(MainActivity.this,PhotoUser.class);
startActivity(intent);
} else
{
//  If sign in fails, display a message to the user.
Toast.makeText(MainActivity.this,"Authentication failed.",
Toast.LENGTH_SHORT).show();
}
}
});
```

Приклад коду реєстрації:

```
mAuth.createUserWithEmailAndPassword (email, password)
.addOnCompleteListener(this, new OnCompleteListener<AuthResult>()
{
@Override
public void onComplete(@NonNull Task<AuthResult> task)
{ if (task.isSuccessful())
 {
Toast.makeText(NextRegistration.this,
"Registation completed.",
Toast.LENGTH_SHORT).show();
} else
{
Toast.makeText(NextRegistration.this, "Error”
Toast.LENGTH_SHORT).show();
}
}
});
```

Для того, щоб зчитувати, або додавати інформацію у базу даних для зручності роботі у проекті необхідно створити клас Java. Основні правила до створення класу:

– в класі повинен бути пустий конструктор;

– ім’я закритих полів класу повинні відповідати ім’ям полів у базі даних;

– для кожного класу повинні бути методи отримання та встановлення значення даних [20].

Окрім цього база даних містить правила доступу, які визначають можливість зчитування та запису даних для користувачів, завдяки чому виключається можливість несанкціонованого доступу до даних (рис. 2.7.).

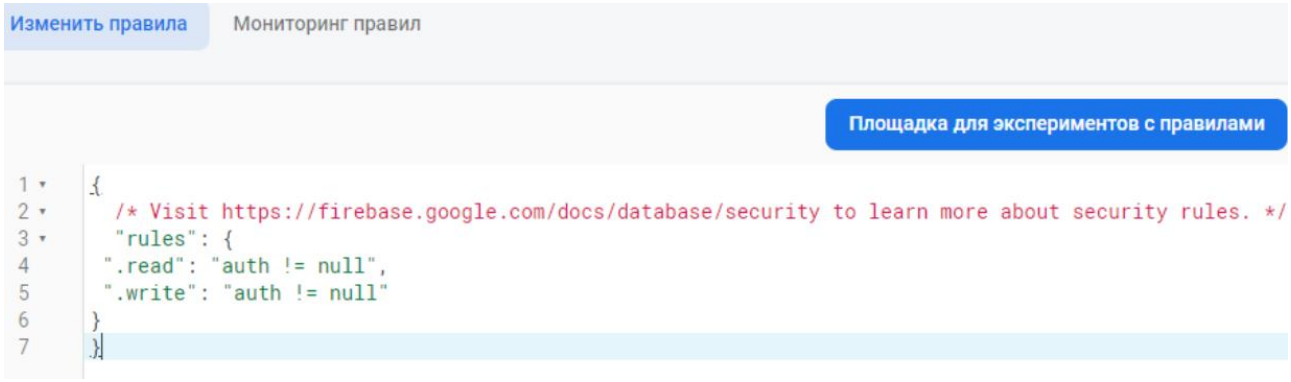


Рис. 2.7. Правила доступу до бази даних

У наведеному прикладі доступ до даних відбувається лише за умови проходження аутентифікації користувачем. В іншому випадку будь-які запити до серверу будуть ігноруватися.

Окрім бази даних сервер FireBase містить хмарне сховище для зберігання файлів з будь-яким розширенням. Окрім файлів воно може містити папки, що покращує організацію даних, а також швидкість доступу. Будь-який файл на сервері містить набір основної інформації, що відображається при натисканні на нього (рис. 2.8 ):

– назва файлу;

– розмір (байт);

– розширення (тип);

– дата створення;

– дата редагування;

– місце знаходження (посилання на файл та його токен доступу);

– додаткові метадані.

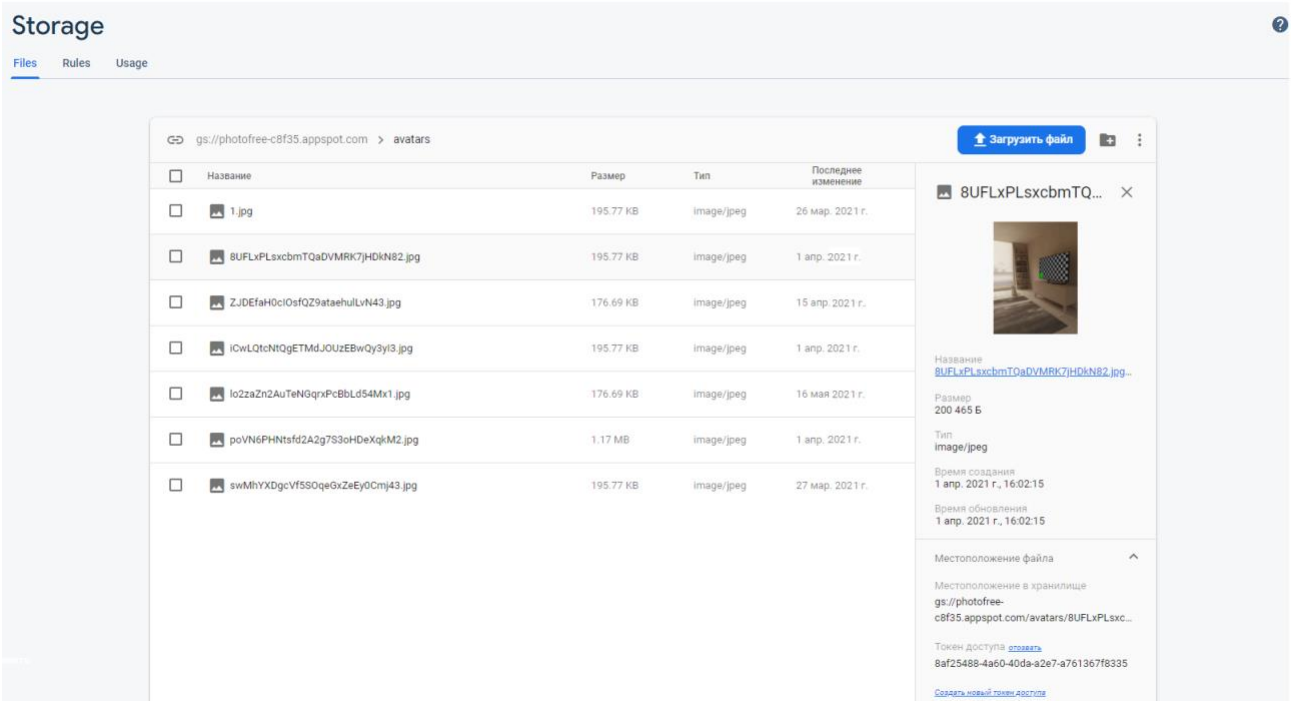


Рис. 2.8. Хмарне сховище серверу Firebase

Окрім безпосередньо самих файлів, сховище містить правила доступу, призначення яких аналогічне базі даних (рис. 2.9).

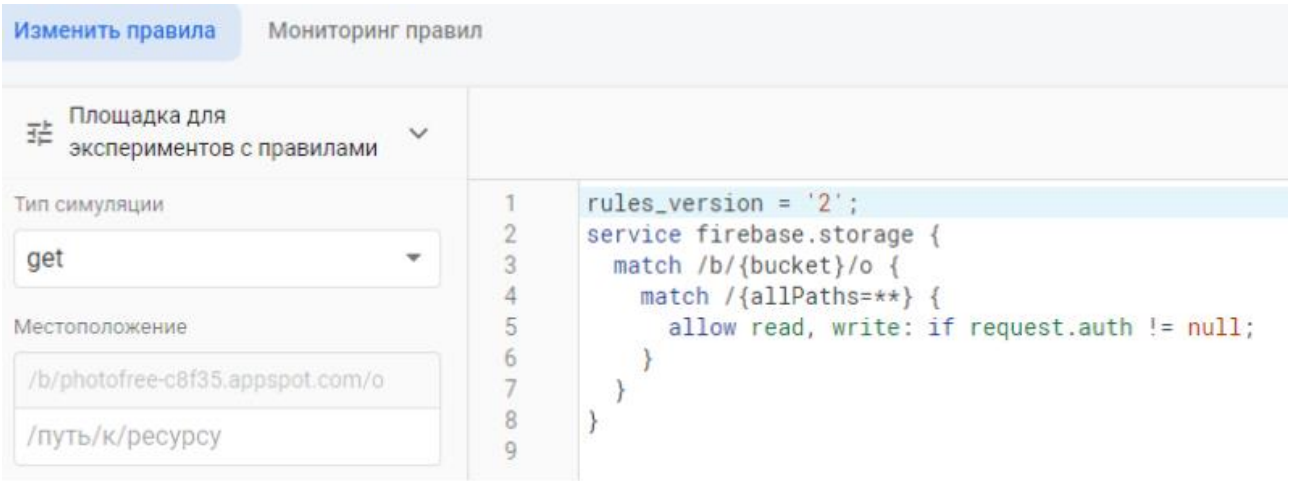


Рис. 2.9. Правила доступу до хмарного сховища

Для додання, або зчитування файлів зі сховища використовується об’єкт StorageReference. При зчитуванні даних необхідно створити тимчасовий файл, в який буде завантажено інформацію.

Приклад коду зчитування даних:

```
StorageReference mStorageRef;
mStorageRef = FirebaseStorage.getInstance().getReference();
StorageReference riversRef = mStorageRef.child("avatars/"+mAuth.getUid()
+".jpg");
File localFile = null;
try {
localFile = File.createTempFile("image", "jpg");
}
catch (IOException e) {
e.printStackTrace();
}
final   String path = localFile.getPath();
riversRef.getFile(localFile).addOnSuccessListener(new OnSuccessListener
<FileDownloadTask.TaskSnapshot>()
{
@Override
public void onSuccess(FileDownloadTask.TaskSnapshot taskSnapshot)
{ //код при успішному завантаженні
}
});
```

Слід відмітити, що створення тимчасового файлу необхідно заключити у блок try-catch, так як існує можливість, що на мобільному пристрої відсутній дозвіл для роботи додатку з файловою системою, або зовнішній накопичувач на має достатньої кількості вільної пам’яті.

Завантаження у сховище виконується аналогічним чином, за винятком того, що у діалоговому вікні обирається необхідний файл, шлях до якого вказується у методі putFile() для об’єкту типу StorageReference.

## 2.4. Опис структури системи та алгоритмів її функціонування

### 2.4.1. Опис структури додатку

#### 2.4.1.1. Архітектура проєкту

Оскільки мобільний додаток передбачає взаємодію користувачів один з одним у реальному часі, то для розробки необхідно використовувати клієнт-серверну архітектуру (рис. 2.10). Вся інформація зберігатиметься на сервері Google Firebase за допомогою бази даних або хмарного сховища. Роботу з даними, а саме обробку та завантаження виконує безпосередньо додаток за умови наявності Інтернет-зв'язку. Підключення серверу до програми виконується за рахунок вже вбудованого плагіну для сервісів Firebase, який додає усі необхідні бібліотеки та посилання.

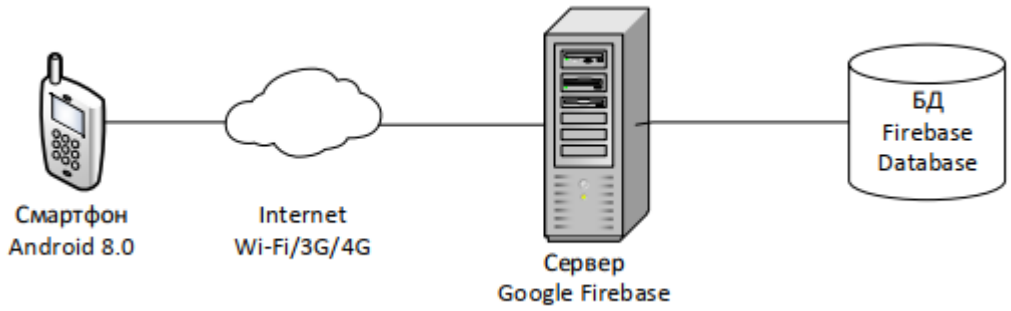


Рис. 2.10. Діаграма розгортання додатку

#### 2.4.1.2. Логічна структура проєкту

Усі користувачі поділяються на 2 категорії (замовники та виконавці), в залежності від якої буде представлено певний функціонал програми (рис. 2.11). Він містить у собі як спільні можливості (наприклад: аутентифікація, ведення персональних даних, налаштування додатку), так і відмінні (наприклад: користувачі категорії «замовники» мають можливість створення та редагування

замовлень, в той час, як «виконавці» взаємодіють лише з існуючими замовленнями).

Інформація про користувачів зберігається на сервері. Для визначення категорії і завантаження відповідної інформації кожен користувач перед початком роботи з програмою проходить аутентифікацію за допомогою електронної адреси та паролю. Якщо обліковий запис відсутній у базі даних, то існує можливість створити новий через проходження реєстрації.

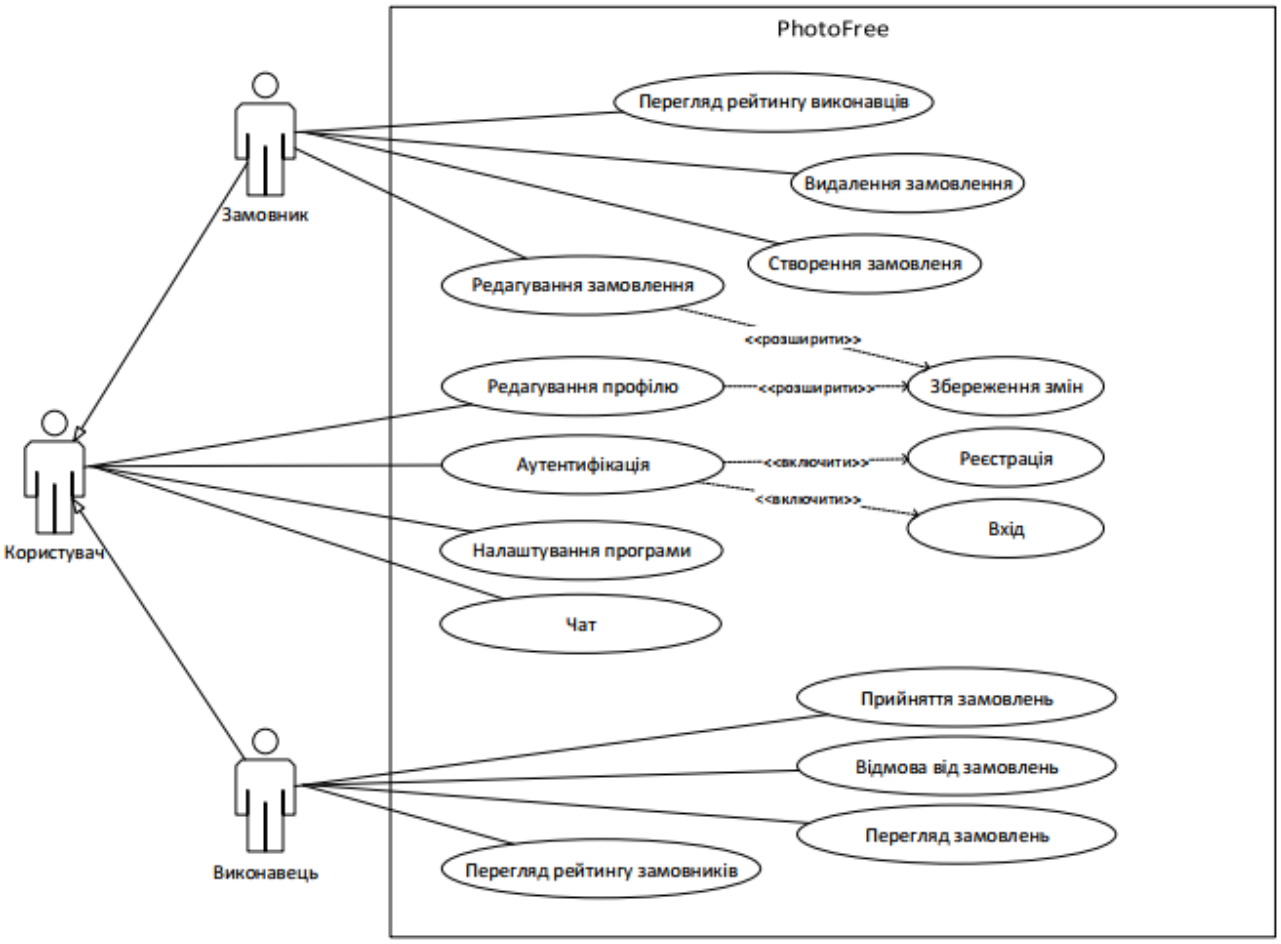


Рис. 2.11. Use Case діаграма

На рис. 2.12 наведено схему співпраці між компонентами системи, на рис. 2.13. наведено контекстну діаграму роботи додатку, а на рис. 2.14 діаграму декомпозиції.


Рис. 2.12. Діаграма співпраці структурних компонентів системи

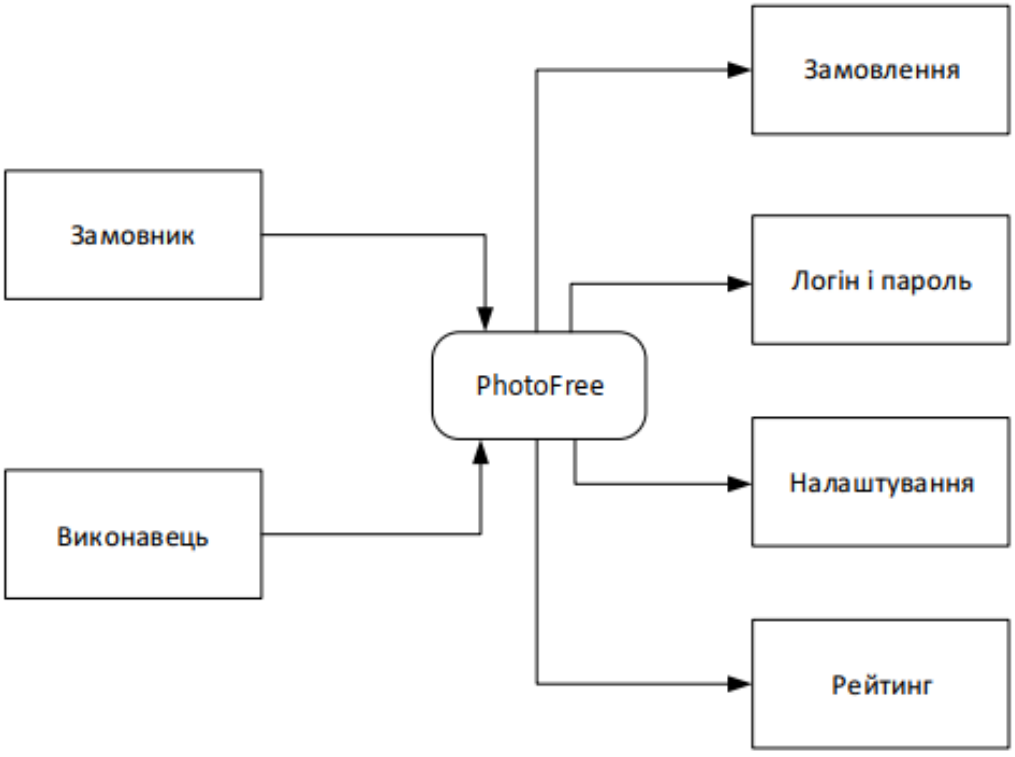


Рис. 2.13. Контекстна діаграма

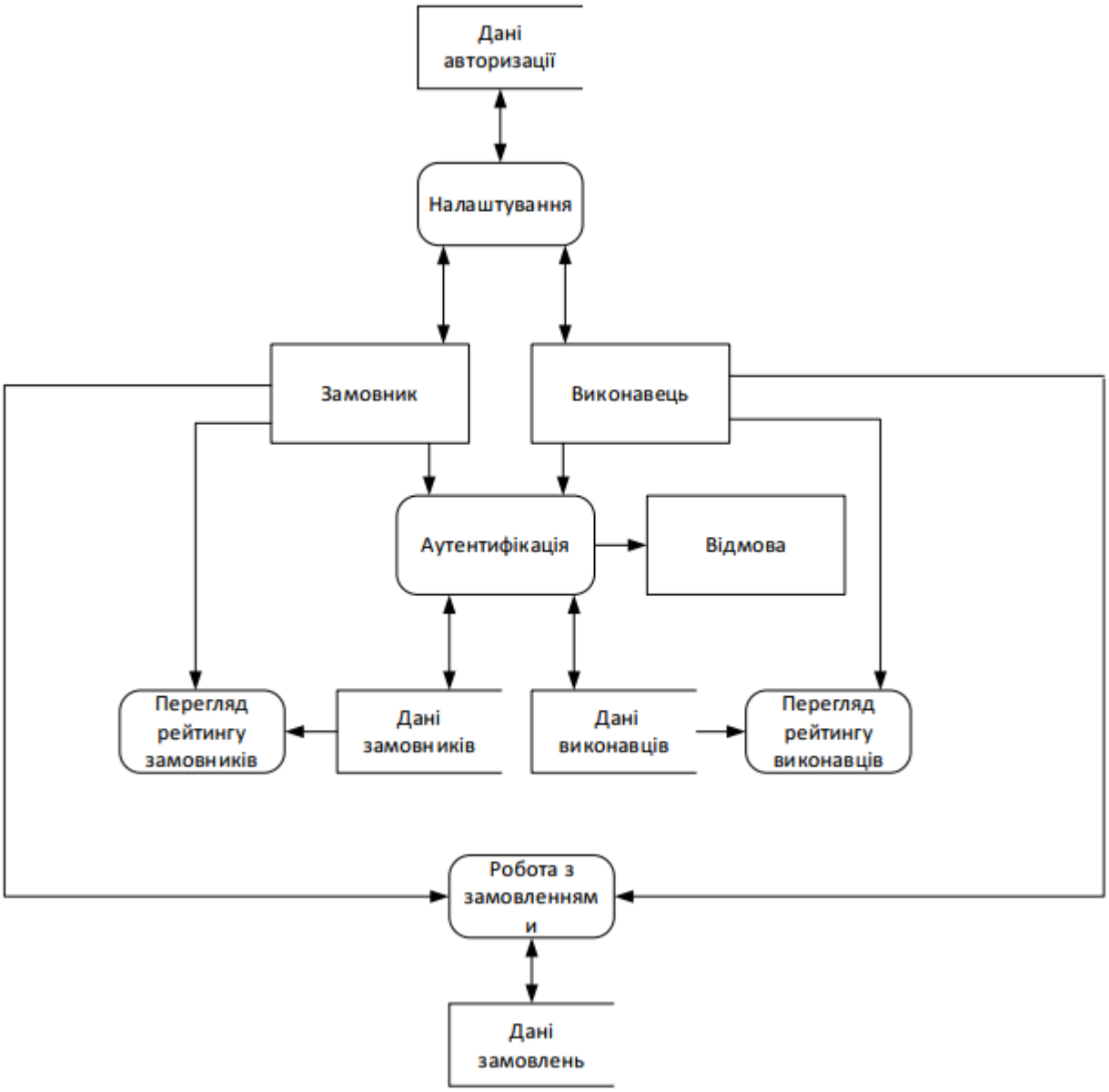


Рис. 2.14. Діаграма декомпозиції

### 2.4.1.3. Файлова структура

При створенні проекту в Android Studio генерується файлова структура з папками:

– Manifest;

– Java;

– Res;

– Gradle Scripts [16].

Папка Manifest містить лише один файл “AndroidManifest.xml”, в якому за допомогою мови XML описуються основні параметри графічного інтерфейсу додатку, а також усіх його вікон.

Папка Java призначена для розміщення усіх класів додатку у файлах з розширенням “.java”. Для більшої зручності роботи її зміст був розбитий на підпапки:

– Custom_classes – користувацькі класи, призначені для роботи з базою даних;

– ui – класи для фрагментів бічного меню та додаткових вікон;

– Views – решта класів, призначених для основних представлень.

Папка res містить файли на мові XML для графічних елементів. Стандартні підпапки:

– Drawable – це файли, які описують користувацьке відображення будь-яких елементів. За допомогою них можна визначити шаблон оформлення окремих об’єктів (наприклад: текстових полів, кнопок, блоків) та застосовувати його необмежену кількість разів;

– Font – містить опис шрифтів, які застосовуються в проекті. Вони можуть бути з розширеннями “.xml” та “.ttf”;

– Layout – це папка з основними файлами відображення графічного інтерфейсу вікон. Окрім цього вона містить опис користувацьких стилів для списків або діалогових вікон;

– Menu – містить лише два файли, за рахунок яких конструюється шаблон бічного меню. Файл “activity_menu_drawer.xml” описує елементи списку кнопок, які розташовані в меню, а файл “buttons_menu.xml” описує випадаюче вікно для налаштувань;

– Mipmap – містить графічні зображення (картинки), що застосовуються в проекті. Мають розширення “.mdpi”, “xxxhdpi”. При імпортуванні зображень типу “.png” або “.jpg” вони автоматично конвертуються у необхідний;

– Navigation – містить лише один файл з картою бічного меню, який відображає графічний інтерфейс кожного фрагменту для зручної навігації між

ними;

– Values – містить графічні константи проекту, наприклад: текст, кольори, стилі, списки. Завдяки зберіганню констант компоненти, що містять однакові властивості, можуть мати лише посилання на константу, а не сам об’єкт.

Папка Gradle Scripts містить систему автоматичної збірки Gradle, яка допомагає підключати зовнішні бібліотеки до проекту, а також налаштовувати конфігурацію для проекту [16].

На рис. 2.15. наведена схема взаємодії компонентів системи.

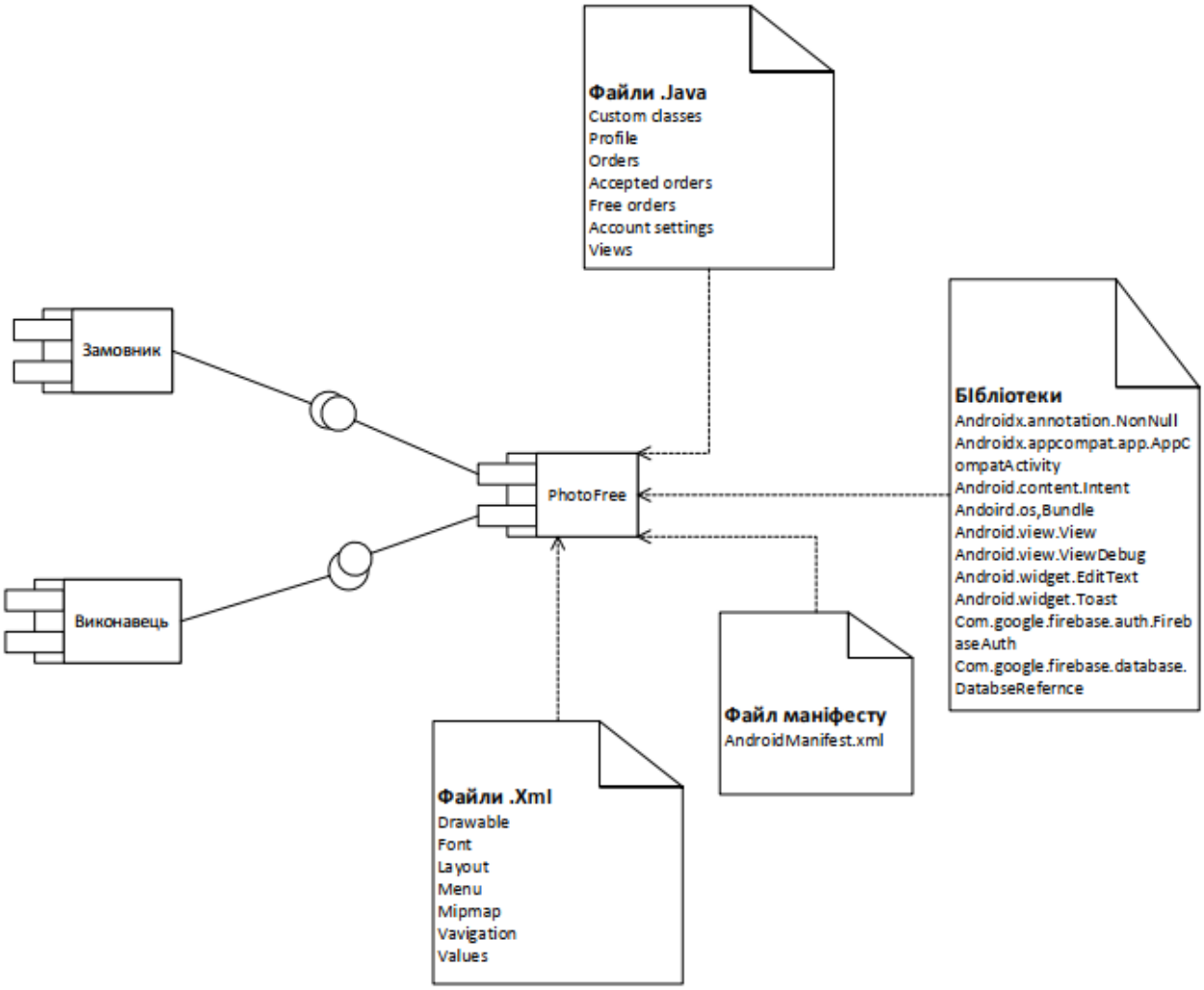


Рис. 2.15. Схема взаємодії компонентів системи

### 2.4.2. Розробка графічного дизайну системи

Інтерфейс мобільного додатку складається з вікон або представлень (View), де розташовані необхідні візуальні та невізуальні компоненти для введення, виведення і обробки даних. Кожне представлення складається з двох файлів: XML, що конструює зовнішній інтерфейс та Java клас, в якому описуються змінні, функції, події, тощо[4].

Вкладки бічного меню складаються с фрагментів (Fragment), що аналогічні представленням, але мають прив’язку до одного класу, за допомогою якого може відбуватися їх обробка.

За рахунок особливості сервера Firebase оновлення інформації у вікні відбувається відразу після внесення змін у базу даних, без необхідності будь-яких дій зі сторони користувача.

При відкритті програми з’являється перше вікно, в якому пропонується авторизуватися за допомогою  пошти та паролю  або пройти реєстрацію (рис. 2.16). Представлення містить візуальні та невізуальні компоненти з рядом властивостей (табл. 2.1).


Рис. 2.16. Вікно авторизації

Вікно містить такі візуальні компоненти:

– заголовок “PhotoFree”;

– вертикальна сітка з полями для введення електронної адреси та паролю;

– горизонтальна сітка з кнопками входу та реєстрації;

Таблиця 2.1

**Властивості компонентів вікна авторизації**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| textView5 | Заголовок з назвою додатку | Text = PhotoFree<br>textColor = #000<br>textSize = 60sp<br>textStyle = bold<br>fontFamily = sans-serif |
| linearLayout | Сітка з полями вводу | Width = 400dp;<br>Orientation = vertical |
| emailEditText | Поле для вводу електронної адреси | Width = 300dp<br>Height = 50dp<br>Background=@drawable/tw_style<br>Hint = Email<br>inputType = textEmailAddress<br>paddingLeft = 10sp<br>textColor = #000<br>textColorHint = #A30<br>fontFamily = roboto_thin |
| passEditText | Поле для вводу пароля | Width = 300dp<br>Height = 50dp<br>Background=@drawable/tw_style<br>Hint = Password<br>inputType = textPassword<br>paddingLeft = 10sp<br>textColor = #000<br>textColorHint = #A30<br>fontFamily = roboto_thin |
| Space | Відступ | Height = 15dp |
| LinearLayout | Сітка з кнопками входу та реєстрації | Orientation = horizontal |

Продовж. табл. 2.1

| Назва компоненту | Опис | Властивості |
|---|---|---|
| loginButton | Кнопка для авторизації | Width = 130 dp<br>Height = 50dp<br>Background=<br>@drawable/tw_style_button<br>Text = Sign in<br>textColor = #FFF<br>textStyle = bold<br>fontFamily = roboto_thin |
| registButton | Кнопка для переходу на реєстрацію | Width = 130 dp<br>Height = 50dp<br>Background=<br>@drawable/tw_style_button<br>Text = Registration<br>textColor = #FFF<br>textStyle = bold<br>fontFamily = roboto_thin |

Якщо користувач не має облікового запису, то при натисканні на кнопку реєстрації відкривається нове вікно для введення основних персональних даних (рис. 2.17). Представлення містить візуальні та невізуальні компоненти з рядом властивостей (табл. 2.2)


Рис. 2.17. Вікно реєстрації

Вікно містить такі візуальні компоненти:

– заголовок “Registration”;

– вертикальна сітка з полями для введення даних;

– текстові поля та їх підписи;

– сітки для поєднання блоку поля і підпису, а також відокремлення від інших блоків;

– кнопка для виклику діалогового вікна вибору міста;

– група перемикачів для вибору категорії користувача;

– кнопка для переходу на наступний етап реєстрації.

Таблиця 2.2

**Властивості компонентів вікна реєстрації**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| headerRegistration | Заголовок реєстрації | Text = Registation<br>textColor = #000<br>textSize = 36sp<br>textStyle = bold<br>fontFamily = roboto_thin<br>gravity = center<br>background=<br>@drawable/shadowtitle |
| linearLayout11 | Сітка з полями для введення даних | Orientation = vertical |
| LinearLayout | Сітки для поєднання блоку поля і підпису | Orientation = vertical<br>Background=<br>@drawable/shadowlinear |
| textView | Заголовки полів вводу | Text = Name, Surname, Phone, City<br>textColor = #9F0<br>textSize = 14sp<br>width = 300dp |
| Button7 | Кнопка для вибору міста | Text = Choose<br>textColor = #000<br>background=<br>@drawable/shadowlinear<br>onClick = OnClickCity |

Продовж. табл. 2.2

| Назва компоненту | Опис | Властивості |
|---|---|---|
| cityEditText | Відображення обраного міста | textColor = #000<br>textSize = 18sp<br>enabled = false |
| radioGroup4 | Група перемикачів для обрання категорії | Background=<br>@drawable/shadowlinear |
| textView6 | Заголовок групи | Text = Type<br>textColor = #70EF<br>textSize = 18sp |
| radioButton1,<br>radioButton2 | Перемикачі категорії | Text = Photographer, Client<br>textColor = #000 |
| registButton | Кнопка для переходу на наступний етап реєстрації | Text = Next<br>textColor = #FFF<br>Height = 35dp<br>Background=<br>@drawable/tw_style_button<br>onClick = onClick |

Діалогове вікно для обрання міста реєстрації (рис 2.18). створюється за допомогою програмного коду та складається з поля для пошуку і списку усіх міст України, які завантажуються з файлу формату XML.

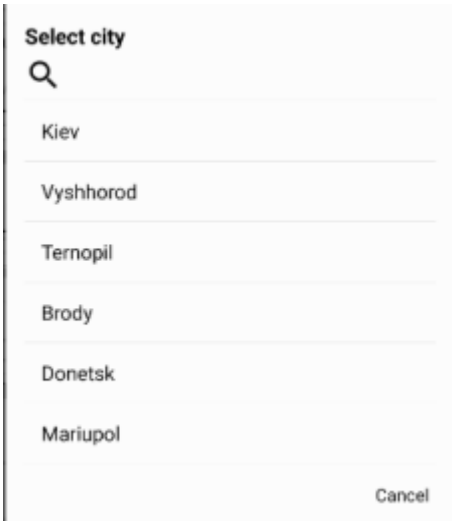


Рис. 2.18. Діалогове вікно для вибору міста

При натисканні на кнопку для переходу та успішній перевірці коректності введених даних відкривається другий етап реєстрації (рис. 2.19) для

введення електронної пошти та створення паролю.

Можливості сервера Firebase дозволяють не зберігати пароль у явному вигляді в базі даних. Це сприяє покращенню безпеки доступу до облікового запису користувача, адже тільки він володітиме значенням власного паролю.

Також на цьому етапі обирається зображення для графічного відображення користувача. Файл картинки обирається з галереї накопичувача мобільного пристрою з допомогою діалогового вікна файлового менеджеру. Після завершення реєстрації зображення зберігається на сервері за допомогою хмарного сховища у папці “avatar” з ім’ям ідентифікатора користувача. Вікно містить такі візуальні компоненти:

– заголовок “Registration”;
– вертикальна сітка з полями для введення даних;
– текстові поля та їх підписи;
– сітки для поєднання блоку поля і підпису, а також відокремлення від інших блоків;
– кнопка для виклику діалогового вікна вибору зображення;
– компонент для зображень;
– кнопка для завершення реєстрації.

Заради покращення зовнішнього виду графічного інтерфейсу замість стандартного компонента відображення картинок, що має квадратну форму, у файл збірки було додано нову бібліотеку “de.hdodenhof:circleimageview:3.1.0”. Вона дозволяє відображати зображення у круглій формі.

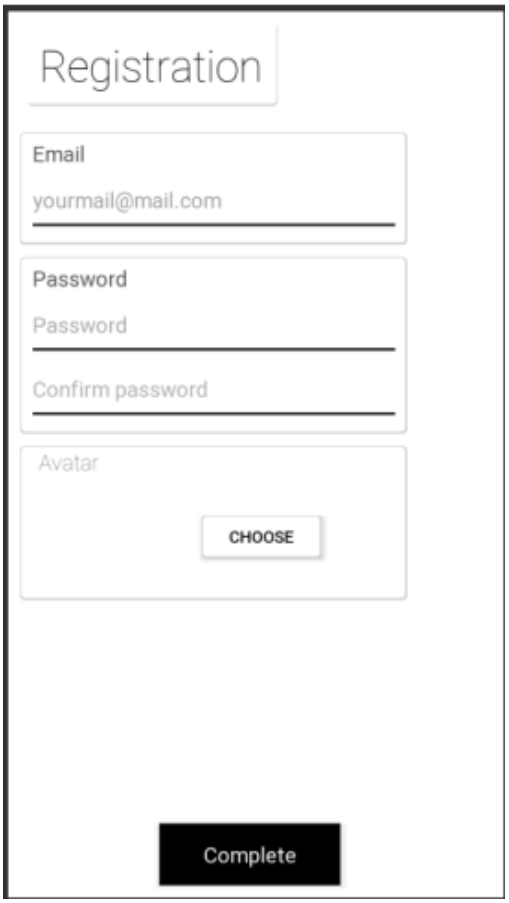


Рис. 2.19. Вікно другого етапу реєстрації

Представлення має візуальні та невізуальні компоненти з рядом властивостей (табл. 2.3).

Таблиця 2.3

**Властивості компонентів вікна другого етапу реєстрації**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| headReg | Заголовок реєстрації | Text = Registation<br>textColor = #000<br>textSize = 36sp<br>textStyle = bold<br>fontFamily = roboto_thin<br>gravity = center<br>background=<br>@drawable/shadowtitle |
| linearLayout12 | Сітка з полями для введення даних | Orientation = vertical |

Продовж. табл. 2.3

| Назва компоненту | Опис | Властивості |
|---|---|---|
| LinearLayout | Сітки для поєднання блоку поля і підпису | Orientation = vertical<br>Background= @drawable/shadowlinear |
| textView | Заголовки полів вводу | Text = Email, Password<br>textColor = #9F0<br>textSize = 14sp<br>width = 300dp |
| mailEditText | Поле вводу електронної адреси | Width = 300dp<br>Height = 43dp<br>Hint = yourmail@gmail.com<br>Background= @drawable/bar_style<br>inputType = textEmailAddress<br>textColor = #000 |
| passEditText, repassEditText | Поля для введення та підтвердження паролю | Width = 300dp<br>Height = 43dp<br>Hint = Password, Confirm password<br>Background= @drawable/bar_style<br>inputType = textPassword<br>textColor = #000 |
| imageView3 | Відображення картинки | Width = height = 80dp<br>Gravity = Center |
| registButton3 | Кнопка для обрання зображення | Text = Choose<br>textColor = #000<br>background= @drawable/shadowlinear<br>onClick = OnClickChoosePhoto |
| registButton2 | Кнопка для завершення реєстрації | Text = Complete<br>textColor = #FFF<br>Height = 35dp<br>Background= @drawable/tw_style_button<br>onClick = onClick |

Після проходження аутентифікації або створення нового облікового запису відкривається профіль користувача та з’являться можливість навігації за допомогою бічного меню, кнопки якого відрізняються в залежності від категорії користувача.

У верхній частині вікон з’являється шапка, на якій розташовані кнопка

для виклику бічного меню та назва поточного фрагменту (рис. 2.20). Шапка міститься в кожному фрагменті бічного меню, але відсутня в додаткових представленнях.


Рис. 2.20. Шапка

Профіль містить усю введену інформацію персональних даних, рейтинг, а також кнопку для переходу у вікно редагування (рис. 2.21). Представлення має візуальні та невізуальні компоненти з рядом властивостей (табл. 2.4).

Вікно містить такі візуальні компоненти:

– текст категорії користувача;
– компонент для зображень;
– текстові поля з персональними даними та їх підписи;
– сітки для поєднання блоку поля і підпису, а також відокремлення від інших блоків;
– кнопка для переходу у вікно редагування даних;
– кнопка для переходу у вікно з рейтингом користувача.

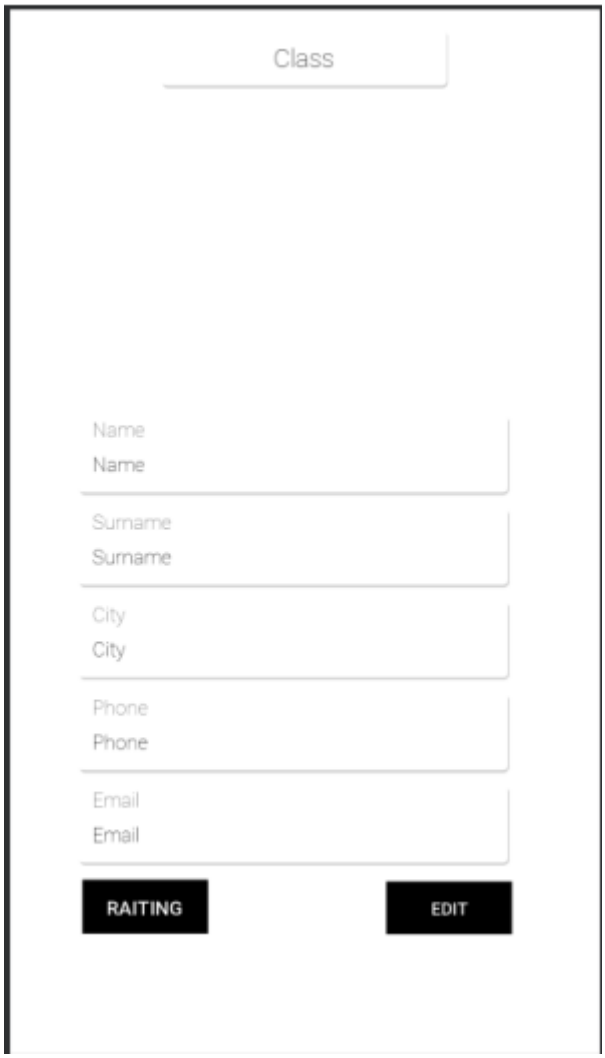


Рис. 2.21. Вікно профілю користувача

Таблиця 2.4

**Властивості компонентів вікна профілю користувача**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| textView11 | Назва категорії користувача | Text = Class (Photographer, Client)<br>textColor = #000<br>textSize = 18sp<br>textStyle = bold<br>fontFamily = roboto_thin<br>gravity = center<br>background= @drawable/shadowtitle |
| avatarMain | Компонент для зображень | Width = Height = 200dp<br>Civ_border_color = #FF0 |

Продовж. табл. 2.4

| Назва компоненту | Опис | Властивості |
|---|---|---|
| LinearLayout | Сітки для поєднання блоку поля і підпису | Orientation = vertical<br>Background=<br>@drawable/shadowtitle |
| textView | Заголовки полів виводу | Text = Name, Surname, City, Phone, Email<br>textColor = #9E0<br>textStyle = bold<br>textSize = 14sp<br>fontFamily = roboto_thin |
| LinearLayout | Сітка для поєднання кнопок | Width = 300dp<br>Orientation = horizontal |
| Button11, Button12 | Кнопку для переходу до рейтингу та редагування профілю | Height = 38dp<br>Text = Raiting, Edit<br>textSize = 12sp<br>background=<br>@drawable/tw_style_button |
| Space | Відстань між кнопками | Width = match_parent |

Вікно редагування персональних даних містить можливість редагування зображення користувача, персональних даних, а також кнопку для збереження змін (рис. 2.22). При її натисканні відредагована інформація завантажується на сервер та відновлюється у полях профілю. Представлення має візуальні та невізуальні компоненти з рядом властивостей (табл. 2.5).

Вікно містить такі візуальні компоненти:

– заголовок “Change ”;

– вертикальна сітка з полями для введення даних;

– текстові поля та їх підписи;

– сітки для поєднання блоку поля і підпису, а також відокремлення від інших блоків;

– компонент для зображень;

– кнопка для виклику діалогового вікна вибору зображення;

– кнопка для виклику діалогового вікна вибору міста;

– кнопка для збереження змін.


Рис. 2.22. Вікно редагування профілю

Таблиця 2.5

**Властивості компонентів вікна редагування профілю**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| changeHead | Заголовок редагування | Text = Change<br>textColor = #000<br>textSize = 24sp<br>width = 233dp<br>fontFamily = roboto_thin<br>gravity = center<br>background=<br>@drawable/shadowtitle |
| linearLayout | Сітка з полями для введення даних | Orientation = vertical |

Продовж. табл. 2.5

| Назва компоненту | Опис | Властивості |
|---|---|---|
| imageButton3 | Кнопка для вибору зображення | Text = Change<br>textColor = #000<br>background=<br>@drawable/shadowlinear<br>onClick = OnClickChoosePhoto |
| Button7 | Кнопка для вибору міста | Text = Choose<br>textColor = #000<br>background=<br>@drawable/shadowlinear<br>onClick = OnClickCity |
| textView | Заголовки полів вводу | Text = Name, Surname, Phone, City<br>textColor = #9F0<br>textSize = 14sp<br>width = 300dp |
| nameEditText, surnameEditText, Phone, City | Поля для введення даних | Width = 300dp<br>Height = 43dp<br>Background=<br>@drawable/bar_style<br>textColor = #000 |
| cityEditText | Відображення обраного міста | textColor = #000<br>textSize = 18sp<br>enabled = false |
| LinearLayout | Сітки для поєднання блоку поля і підпису | Orientation = vertical<br>Background=<br>@drawable/shadowlinear |
| saveButton | Кнопка для збереження змін | Text = Next<br>textColor = #FFF<br>Height = 35dp<br>Background=<br>@drawable/tw_style_button<br>onClick = onClick |

Вікно рейтингу користувача містить візуальне відображення оцінок у вигляді зірок від 1 до 5 балів. Навпроти кожного балу відображається кількість користувачів, що поставили дану оцінку. У правій частині відображається середнє значення відносно усіх балів (рис. 2.23). Представлення має візуальні та невізуальні компоненти з рядом властивостей (табл. 2.6).


Рис. 2.23. Вікно рейтингу користувача

Вікно містить такі візуальні компоненти:

– заголовок “Raiting ”;
– вертикальна сітка з рейтингом від 1 до 5 балів;
– заголовок середнього рейтингу “Total”;
– середній рейтинг у графічному та числовому вигляді.

Таблиця 2.6

**Властивості компонентів вікна рейтингу користувача**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| linearLayout17 | Сітка з рейтингом від 1 до 5 балів | Orientation = vertical<br>Background=<br>@drawable/shadowlinear |
| LinearLayout | Сітки для кожного балу | Orientation = horizontal |
| raitingHead | Заголовок рейтингу | Text = Raiting<br>textColor = #000<br>textSize = 24sp<br>width = 233dp<br>fontFamily = roboto_thin<br>gravity = center<br>background=<br>@drawable/shadowtitle |
| textView2 | Заголовок середнього значення рейтингу | Text = Total<br>textColor = #000<br>textSize = 36sp |
| textView (44-48) | Візуальне представлення кожного балу рейтингу | Text = ☆☆☆☆☆<br>textColor = #000<br>textSize = 20sp |
| textView (49-53) | Кількість оцінок для кожного балу рейтингу | Text = 0<br>textColor = #000<br>textSize = 20sp |
| textView54 | Середній бал рейтингу у графічному вигляді | Text = ☆☆☆☆☆<br>textColor = #000<br>textSize = 36sp |
| textView55 | Середній бал рейтингу у числовому вигляді | Text = 0<br>Gravity = center \| top<br>textColor = #000<br>textSize = 36sp |

Інформація про замовлення знаходиться в окремому пункті меню навігації під назвою “Orders” як для замовників, так і для виконавців. Вікно містить список ListView, елементи відображення якого було реалізовано власноруч в окремому файлі “order_list_element.xml” (рис. 2.24).

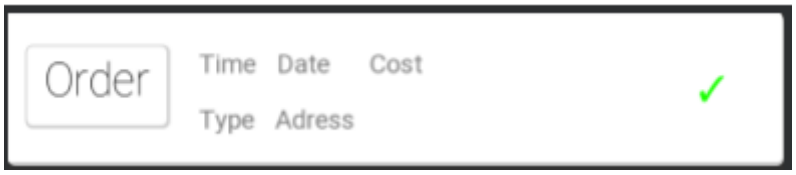


Рис. 2.24. Елемент списку

Елемент містить такі візуальні компоненти:

– заголовок “Order ”;

– підписи для відображення інформації про замовлення;

– статус замовлення у вигляді символу;

Представлення має візуальні та невізуальні компоненти з рядом властивостей (табл. 2.7).

Таблиця 2.7

**Властивості компонентів елементу списку замовлень**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| elOrderTV | Заголовок елементу | Text = Order<br>textColor = #000<br>textSize = 36sp<br>textStyle = bold<br>fontFamily = roboto_thin<br>background=<br>@drawable/shadowtitle |
| LinearLayout | Головна сітка, що розділяє заголовок та інформацію | Orientation = horizontal |
| LinearLayout | Сітка, що розділяє інформацію на два рядки | Orientation = vertical |
| elTimeTV, elDateTV, elCostTV, elTypeTV, elAdressTV | Відображення інформації про замовлення | textColor = #81000<br>textSize = 14sp |
| elStatusTV | Відображення статусу замовлення | textColor = #1AFF01<br>textSize = 18sp<br>gravity = center \| right |

Окрім списку вікно містить кнопку, для додавання нового замовлення

(рис. 2.25). При її натисканні відкривається форма введення даних (рис. 2.226).

Рис. 2.25. Кнопка додання нового замовлення


Рис. 2.26. Вікно додання замовлення

Вікно містить такі візуальні компоненти:

– заголовок “Order ”;
– сітки для відокремлення даних один від одного;
– кнопка для відкриття діалогового вікна вибору дати;
– кнопка для відкриття діалогового вікна вибору часу;
– випадаючий список вибору типу замовлення;

– кнопка для відкриття діалогового вікна вибору міста;
– кнопка для відкриття діалогового вікна вибору тривалості;
– поля введення адреси, ціни та опису;
– кнопка для підтвердження додання замовлення.

Діалогові вікна вибору дати та часу використовують стандартні шаблони для операційної системи Android у вигляді календаря та годинника відповідно (рис. 2.27, 2.28). Вибір тривалості представляє собою прокрутку чисел від 1 до 24 (рис. 2.29).

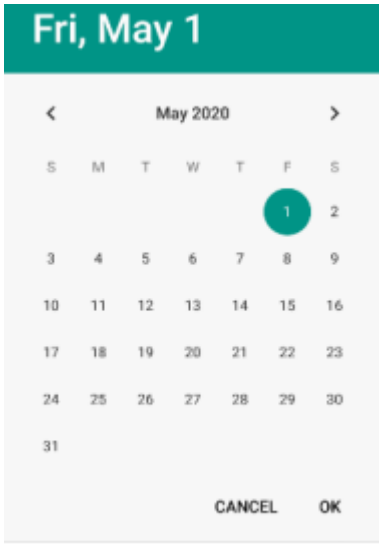


Рис. 2.27. Діалогове вікно вибору дати

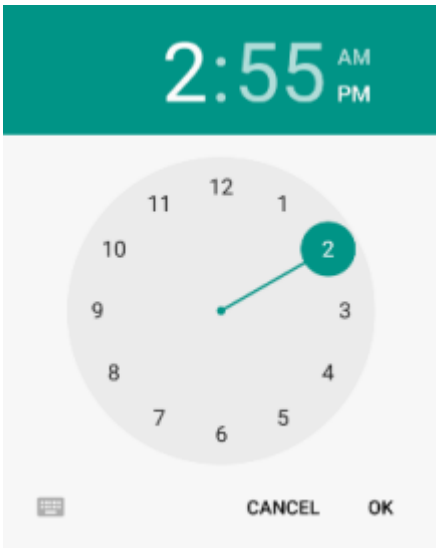


Рис. 2.28. Діалогове вікно вибору часу


Рис. 2.29. Діалогове вікно вибору тривалості замовлення

Представлення має візуальні та невізуальні компоненти з рядом властивостей (табл. 2.8).

Таблиця 2.8

**Властивості компонентів вікна додання замовлення**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| textView | Заголовок елементу | Text = Order<br>textColor = #000<br>textSize = 24sp<br>textStyle = bold<br>background=<br>@drawable/shadowtitle |
| textView (1-7) | Інформаційні підписи | Text = Order<br>textColor = #000<br>textSize = 14sp |
| chooseDate | Кнопка відкриття діалогового вікна вибору дати | background=<br>@drawable/shadowtitle<br>onClick = OnClickDate<br>text = Choose<br>textSize = 12sp<br>textColor = #000 |
| chooseTime | Кнопка відкриття діалогового вікна вибору часу | background=<br>@drawable/shadowtitle<br>onClick = OnClickTime<br>textSize = 12sp<br>textColor = #000 |

Продовж. табл. 2.8

| Назва компоненту | Опис | Властивості |
|---|---|---|
| chooseDuration | Кнопка відкриття діалогового вікна вибору тривалості | background= @drawable/shadowtitle<br>onClick = OnClickDuration<br>text = Choose<br>textSize = 12sp<br>textColor = #000 |
| spinner | Випадаючий список вибору типу | spinnerMode = dropdown<br>textColor = #000 |
| Button4 | Кнопка відкриття діалогового вікна вибору міста | background= @drawable/shadowtitle<br>onClick = OnClickCity<br>text = City<br>textSize = 12sp<br>textColor = #000 |
| linearLayout | Сітки для відокремлення даних | background= @drawable/shadowlinear<br>orientation = horizontal |
| adressEditText | Поле введення адреси | hint = Street/Place<br>textColor = #000<br>textSize = 12sp<br>background= @drawable/bar_style |
| costEditText | Поле введення ціни | textColor = #000<br>textSize = 12sp<br>background= @drawable/bar_style |
| desEditText | Поле введення опису | textColor = #000<br>inputType = textMultiLine<br>textSize = 12sp |
| button | Додання нового замовлення | background= @drawable/tw_style_button<br>onClick = OnClickAdd<br>text = Add<br>textColor = #000<br>textSize = 12sp |

Після додання нового замовлення вікно автоматично закривається, а також виконується автоматичне оновлення списку замовлень. Щоб переглянути деталі обраного замовлення необхідно натиснути на нього, після чого

відкривається вікно з детальною інформацією (рис. 2.30). Окрім даних вікно містить кнопки для редагування, видалення, підтвердження, а також перегляду списку виконавців, що приєдналися до обраного замовлення. Якщо замовник вже обрав потенційного виконавця, то кнопка відкриватиме його профіль, а не список.

Редагування відбувається у вікні, що аналогічне вікну додання, за винятком того, що перше містить вже введену інформацію, яку можливо змінити. Кнопка підтвердження змінює статус замовлення на виконане, але є доступною лише за наявності обраного виконавця.

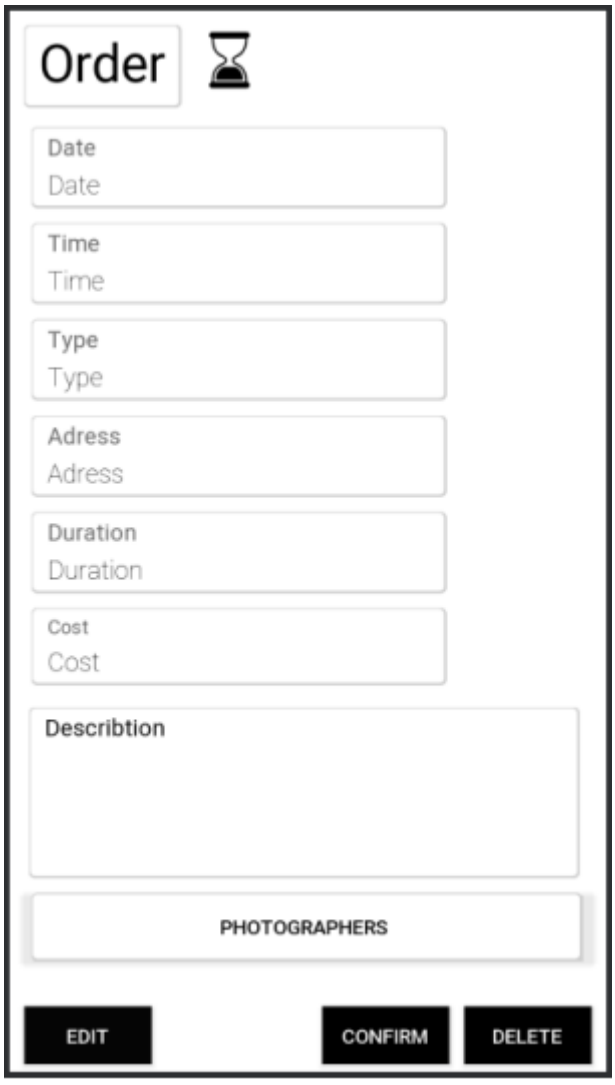


Рис. 2.30. Вікно перегляду деталей замовлення

Вікно містить такі візуальні компоненти:

– заголовок “Order ”;

– відображення поточного статусу замовлення;

– сітки для відокремлення даних один від одного;

– окремо виділений опис замовлення;

– кнопка відкриття редагування;

– кнопка видалення;

– кнопка підтвердження;

– кнопка перегляду списку або обраного виконавця.

Представлення має візуальні та невізуальні компоненти з рядом властивостей (табл. 2.9).

Таблиця 2.9

**Властивості компонентів вікна додання замовлення**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| textView28 | Заголовок елементу | Text = Order<br>textColor = #000<br>textSize = 36sp<br>textStyle = bold<br>background=<br>@drawable/shadowtitle |
| textView15 | Статус замовлення | textSize = 36sp<br>textColor = #000 |
| linearLayout | сітки відокремлення даних один від одного | background=<br>@drawable/shadowlinear<br>orientation = vertical |
| textView (1-7) | Інформаційні підписи | textColor = #990000<br>textSize = 16sp |
| textView (8-14) | Дані про замовлення | textColor = #000<br>textStyle = bold<br>textSize = 18sp<br>fontFamily = roboto_thin |
| Button9 | Кнопка перегляду списку або обраного виконавця | Text = Photographers<br>textColor = #000<br>onClick = OnClickPhotographers<br>background=<br>@drawable/shadowlinear |

Продовж. табл. 2.9

| Назва компоненту | Опис | Властивості |
|---|---|---|
| Button3 | Кнопка редагування | Text = Edit<br>onClick = OnClickChangeOrder<br>background = #050505 |
| Button6 | Кнопка видалення | Text = Delete<br>onClick = OnClickDeleteOrder<br>background = #050505 |
| Button5 | Кнопка підвердження | Text =Confirm<br>onClick = OnClickConfirmOrder<br>background = #050505 |

Вікно списку виконавців містить ListView, елементи відображення якого було реалізовано власноруч в окремому файлі “photographer_list_element.xml” (рис. 2.31). Елемент має візуальні та невізуальні компоненти з рядом властивостей (табл. 2.10).


Рис. 2.31. Елемент списку виконавців

Елемент містить такі візуальні компоненти:

– зображення профілю користувача;
– текстове поле для прізвища та ім’я;
– середній рейтинг у графічному вигляді.

Таблиця 2.10

**Властивості компонентів елементу списку виконавців**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| cirleImageView | Зображення профілю користувача | Width = 80dp<br>Height = 80dp |
| NameSurname | Текстове поле для прізвища та ім’я | textColor = #000<br>textSize = 14sp |
| Raiting | Середній рейтинг | textColor = #000<br>textSize = 14sp |

При натисканні на елемент списку відкривається вікно перегляду профілю виконавця (рис. 2.32), де відображається детальна інформація персональних даних (фото, прізвище, ім’я, телефон, електронна адреса) та рейтингу (інформація про рейтинг аналогічна тій, що знаходиться на вкладці профілю користувачів).

Для прийняття виконавця до замовлення у вікні знаходиться відповідна кнопка. У випадку, якщо він уже прийнятий до замовлення, при натисканні на кнопку відбудеться обернена дія (прийняття виконавця буде відмінено). Також вікно має кнопку відкриття чату, де існує можливість зв’язатися с користувачем.

Вікно містить такі візуальні компоненти:

– зображення профілю користувача;
– сітка з основною інформацією;
– дані профілю;
– сітка з рейтингом;
– дані рейтингу.


Рис. 2.32. Вікно перегляду профілю

Представлення має візуальні та невізуальні компоненти з рядом властивостей (табл. 2.11).

Таблиця 2.11

**Властивості компонентів вікна перегляду профілю**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| avatarPhotographer | Зображення профілю | Width = 150dp<br>Height = 150dp<br>Background = #fff |
| linearLayout | Сіткаzосновною інформацією | Orientation = vertical<br>Background=<br>@drawable/shadowlinear |
| linearLayout | Сітка з рейтингом | background=<br>@drawable/shadowlinear<br>orientation = vertical |

Продовж. табл. 2.11

| Назва компоненту | Опис | Властивості |
|---|---|---|
| textView36 | Заголовок основної інформації | textSize = 24sp<br>textColor = #92070707<br>fontFamily = allerta |
| TextView22/34/35 | Поля з основною інформацією | textColor = #000<br>textStyle = bold<br>fontFamily = roboto_thin |
| textView43 | Заголовок рейтингу | textSize = 24sp<br>textColor = #92070707<br>fontFamily = allerta |
| textView2 | Заголовок середнього значення рейтингу | Text = Total<br>textColor = #000<br>textSize = 36sp |
| textView (44-48) | Візуальне представлення кожного балу рейтингу | Text = ☆☆☆☆☆<br>textColor = #000<br>textSize = 14sp |
| textView (49-53) | Кількість оцінок для кожного балу рейтингу | Text = 0<br>textColor = #000<br>textSize = 14sp |
| textView54 | Середній бал рейтингу у графічному вигляді | Text = ☆☆☆☆☆<br>textColor = #000<br>textSize = 30sp |
| textView55 | Середній бал рейтингу у числовому вигляді | Text = 0<br>Gravity = center \| top<br>textColor = #000<br>textSize = 24sp |
| Button8 | Кнопка прийняття виконавця до замовлення | Text = “Accept to order”<br>textColor = #fff<br>background = #000<br>onClick=AcceptOrRemove |
| imageButton2 | Кнопка відкриття чату | srcCompat = @mimap/chat<br>OnClick = OnClickChat |

При натисканні кнопки відкриття чату відкривається нове вікно в якому відображається діалог між замовником та виконавцем (рис. 2.33). Представлення містить набір властивостей (табл. 2.12). У верхній частині відображається прізвище та ім’я співрозмовника. Основну частину займає список listView з користувацькими елементами (рис. 2.34), в яких розташовані

компоненти повідомлень (табл. 2.13). Кожне повідомлення представляє собою кольоровий елемент списку (повідомлення користувача мають синій колір, а повідомлення співрозмовника зелений), що містить текст, час, а також дату відправки. В залежності від відправника повідомлення відображається з лівої або правої сторони екрану. Також вікно містить багаторядкове поле для введення та кнопку для відправки повідомлення. Усі діалоги зберігаються на сервері у базі даних. При повторному відкритті вікна раніше створені повідомлення будуть відображені у списку.

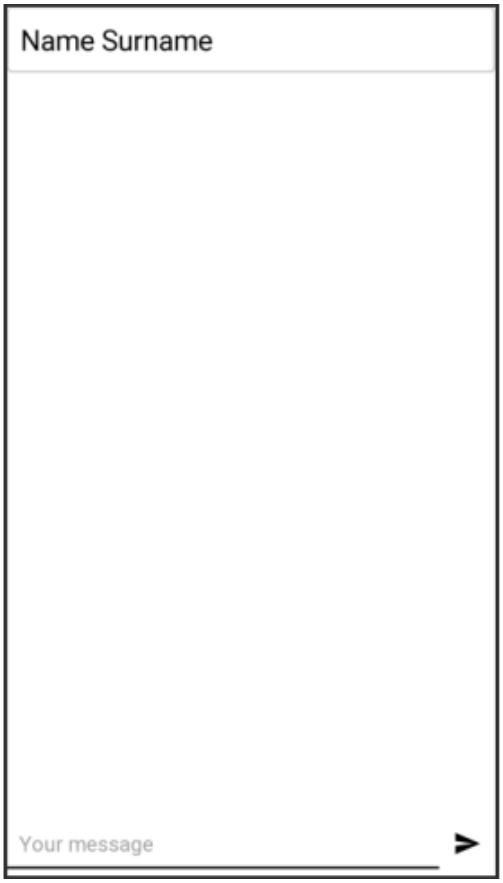


Рис. 2.33. Вікно діалогу

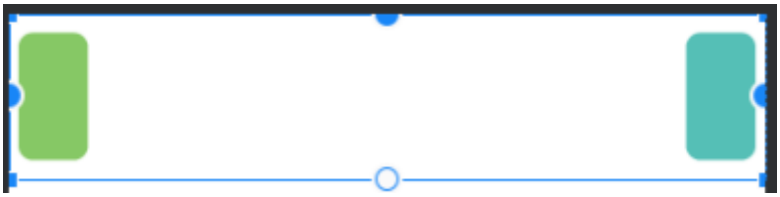

Рис. 2.34. Елемент списку діалогу

Таблиця 2.12

**Властивості компонентів вікна діалогу**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| linearLayout14 | Заголовок вікна | Orientation = horizontal<br>Background=<br>@drawable/shadowlinear |
| textView56 | Прізвище та ім’я співрозмовника | textSize = 24sp<br>textColor = #000 |
| dialogW | Список з повідомленнями | transriptMode = alwaysScroll<br>stackFromBottom = true |
| editText2 | Поле для введення нового повідомлення | Hint = “Your message”<br>Background=<br>@drawable/bar_style<br>inputType = textMultiLine<br>textColor = #000<br>textColorHint = #610000 |
| imageButton | Кнопка для відправлення повідомлення | Background = #00f<br>OnClick = OnClickSendMessage<br>srcComptat=<br>@drawable/ic_menu_send |

Таблиця 2.13

**Властивості компонентів елементу списку діалогу**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| LinearLayout | Сітка для об’єднання компонентів елементу | Orientation = vertical |
| chatLeft | Сітка для повідомлення співрозмовника | Background=<br>@drawable/messag_box_left<br>Orientation = vertical |
| messageTextLeft | Поле тексту повідомлення співрозмовника | inputTime = textMultiLine<br>enabled = false<br>textColor = #000 |
| messageTimeLeft | Час відправлення повідомлення співрозмовника | textColor = #800<br>textSize = 12sp |
| messageDateLeft | Дата відправлення повідомлення співрозмовника | textColor = #800<br>textSize = 12sp |

Продовж. табл. 2.13

| Назва компоненту | Опис | Властивості |
|---|---|---|
| Space | Відступ для повідомлень | Width = match_parent<br>Weight = 1 |
| chatRight | Сітка для повідомлення користувача | Background=<br>@drawable/messag_box_right<br>Orientation = vertical |
| messageTextRight | Поле тексту повідомлення користувача | inputTime = textMultiLine<br>enabled = false<br>textColor = #000 |
| messageTimeRight | Час відправлення повідомлення користувача | textColor = #800<br>textSize = 12sp |
| messageDateRight | Дата відправлення повідомлення користувача | textColor = #800<br>textSize = 12sp |

Якщо користувач має тип виконавця, то замість створення нових замовлень йому доступні функції перегляду та обирання вже створених. В такому разі головне меню містить 2 вкладки: перша містить вже обрані замовлення, друга має список усіх вільних замовлень, а також можливість фільтрації за певними критеріями.

Вікно прийнятих замовлень складається зі списку замовлень, аналогічний списку для замовників, за виключенням того, що замість номеру замовлення на місці розташоване зображення профілю замовника (рис. 2.35).

Представлення має візуальні та невізуальні компоненти з рядом властивостей (табл. 2.14).


Рис. 2.35. Елемент списку замовлень

Елемент містить такі візуальні компоненти:

– зображення профілю замовника;

– підписи для відображення інформації про замовлення;
– вартість замовлення.

Таблиця 2.14

**Властивості компонентів елементу списку замовлень**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| avatarClient | Зображення профілю користувача | Background = #FFF<br>Width = 60dp<br>Height = 60dp |
| LinearLayout | Головна сітка, що розділяє зображення та інформацію | Orientation = horizontal |
| LinearLayout | Сітка, що розділя інформацію на два рядки | Orientation = vertical |
| elTimeTV, elDateTV, elTypeTV, elAdressTV | Відображення інформації про замовлення | textColor = #81000<br>textSize = 14sp |
| elCostTV | Вартість замовлення | textColor = #81000<br>textSize = 24sp |

При натисканні замовлення відкривається вікно з детальною інформацією. Його зовнішній вигляд аналогічний вікну для замовників (рис. 2.30). Основною відмінністю є те, що замість кнопок перегляду виконавців, редагування, підтвердження та видалення розташовані кнопки перегляду профілю замовника, а також прийняття або відмова від замовлення. При натисканні кнопки відмови вікно детальної інформації закривається і замовлення видаляється зі списку прийнятих.

Вікно вільних замовлень складається з аналогічного списку. Додатково воно містить кнопку для відкриття вікна фільтрації замовлень (рис. 2.36). При натисканні на кнопку з’являється нова форма для редагування та застосування фільтру, який виключатиме зайві елементи списку та відображатиме лише ті, що відповідають умові (рис. 2.37).


Рис. 2.36. Кнопка фільтрації вільних замовлень


Рис. 2.37. Вікно фільтрації вільних замовлень

Вікно містить такі візуальні компоненти:

– заголовок “Filter”;

– сітки для відокремлення даних один від одного;

– кнопки для відкриття діалогового вікна вибору початкової та кінцевої дати;

– кнопки для відкриття діалогового вікна вибору раннього та пізнього часу;

– кнопка для відкриття діалогового вікна вибору міста;

– кнопки для відкриття діалогового вікна вибору мінімальної та максимальної тривалості;

– поля для введення мінімальної та максимальної ціни;

– кнопка підтвердження фільтру;

– кнопка очищення фільтру.

Представлення має візуальні та невізуальні компоненти з рядом властивостей (табл. 2.15).

Таблиця 2.15

**Властивості компонентів вікна фільтрації**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| textView | Заголовок елементу | Text = Filter<br>textColor = #000<br>textSize = 24sp<br>textStyle = bold<br>background=<br>@drawable/shadowtitle |
| textView (1-5) | Інформаційні підписи | textColor = #000<br>textSize = 14sp |
| chooseDateFilter1,<br>chooseDateFilter2, | Кнопки відкриття діалогового вікна вибору дати | background=<br>@drawable/shadowtitle<br>onClick = OnClickDateFilter1/<br>onClick = OnClickDateFilter2<br>text = From/to<br>textSize = 12sp<br>textColor = #000 |
| dateFilter1TextView,<br>dateFilter2TextView | Поля відображення обраної дати | textSize = 18sp<br>textColor = #000<br>fontFamily = roboto_thin<br>textStyle = bold |
| chooseTimeFilter1,<br>chooseTimeFilter2 | Кнопка відкриття діалогового вікна вибору часу | background=<br>@drawable/shadowtitle<br>onClick = OnClickTimeFilter1/<br>onClick = OnClickTimeFilter2<br>text = From/To<br>textSize = 12sp<br>textColor = #000 |

Продовж. табл. 2.15

| | | |
|---|---|---|
| timeFilter1TextView, timeFilter2TextView | Поля відображення обраного часу | textSize = 18sp<br>textColor = #000<br>fontFamily = roboto_thin<br>textStyle = bold |
| chooseDurationFilter1, chooseDurationFilter2 | Кнопка відкриття діалогового вікна вибору тривалості | background= @drawable/shadowtitle<br>onClick= OnClickDurationFilter1/<br>onClick = OnClickDurationFilter2<br>text = From/To<br>textSize = 12sp<br>textColor = #000 |
| durationFilter1TextView, durationFilter2TextView | Поля відображення обраної тривалості | textSize = 18sp<br>textColor = #000<br>fontFamily = roboto_thin<br>textStyle = bold |
| chooseCityFilter | Кнопка відкриття Діалогового вікна вибору міста | background= @drawable/shadowtitle<br>onClick = OnClickCityFilter<br>Text = Choose<br>textSize = 12sp<br>textColor = #000 |
| costFilter1EditText, costFilter2EditText | Поля для введення ціни | Background= @drawable/bar_style<br>inputType = numberDecimal<br>textColor = #000<br>textSize = 12sp |

Обидві сторони мають вкладку для налаштування даних аккаунту, в якій існує можливість змінювати пошту або пароль, що використовуються для аутентифікації (рис 2.38). Зміна даних відбувається лише за умови введення поточного паролю у відповідне поле. Окрім цього знизу розташована кнопка для виходу з системи, яка переміщує користувача у вікно авторизації без можливості повернення назад.

Вікно містить такі візуальні компоненти:

– заголовок;
– сітка для об’єднання даних;
– заголовки полів введення;
– поле для введення старого паролю;
– поле для введення нової пошти;
– поле для введення нового паролю;
– кнопки для підтвердження зміни;
– кнопка виходу з системи.

Представлення має візуальні та невізуальні компоненти з рядом властивостей (табл. 2.16).


Рис. 2.28. Вікно налаштувань аккаунту

Таблиця 2.16

**Властивості компонентів вікна фільтрації**

| Назва компоненту | Опис | Властивості |
|---|---|---|
| textView61 | Заголовок елементу | Text = Change data<br>textColor = #000<br>textSize = 24sp<br>textStyle = bold<br>fontFamily = @font/allerta<br>background=<br>@drawable/bar_style |
| linearLayout18 | Сітка об’єднання даних | Orientation = vertical<br>Background=<br>@drawable/shadowlinear |
| textView64/65/66 | Заголовки полів введення | textColor = #000<br>fontFamily = @font/allerta |
| editText6 | Поля введення | background=<br>@drawable/bar_style<br>textColor = #000<br>textColorHint = #6D0 |
| Button22/23 | Кнопки підтвердження зміни | Text = complete<br>textColor = #000<br>background=<br>@drawable/shadowlinear |
| Button18 | Кнопка виходу з системи | Text = Sign out<br>textColor = #000<br>background=<br>@drawable/shadowlinear<br>OnClick = Logout |

### 2.4.3. Проектування бази даних

#### 2.4.3.1. Концептуальне проектування

Предметна область: freelance-платформа для клієнтів та фотографів. База даних повинна містити інформацію про користувачів, їх рейтинг, діалоги та замовлення. Усі користувачі поділяються на дві категорії (замовники та виконавці), а також мають власні персональні дані, такі як: прізвище, ім’я, номер телефону, місце проживання, електронна адреса.

Категорія замовників має можливість створення замовлень, які містять у собі: ідентифікатор замовника, адресу, вартість, дату, час, тривалість, тип, статус, ідентифікатор виконавця, якого було обрано для виконання замовлення. До замовлення можуть підписуватися декілька виконавців, серед яких замовник обирає лише одного.

Усі діалоги прив'язані до конкретного замовлення з виконавцем та складаються з повідомлень, що містять у собі таку інформацію: дата та час відправлення, тип відправника, текст повідомлення. Відправляти та отримувати повідомлення мають можливість обидві сторони.

Після виконання замовлення обидві сторони мають можливість провести оцінювання якості роботи протилежної сторони. Для цього використовується п'ятизіркова рейтингова система. В рейтингу кожного користувача повинна знаходитися інформація про кількість разів оцінювання відносно кожної оцінки (від 1 до 5), за допомогою чого існує можливість розрахунку середньої оцінки.

В результаті проведеного аналізу можна виділити основні сутності для бази даних:

– користувачі. Основні атрибути сутності: ідентифікатор, прізвище, ім'я, номер телефону, місце проживання, електронна адреса, тип користувача;

– замовлення. Основні атрибути сутності: ідентифікатор, ідентифікатор замовника, адреса, вартість, дата, час, тривалість, тип, статус, ідентифікатор виконавця;

– діалог. Основні атрибути сутності: ідентифікатор, ідентифікатор замовлення, ідентифікатор виконавця;

– повідомлення. Основні атрибути сутності: ідентифікатор, дата та час відправлення, тип відправника, текст повідомлення;

– рейтинг. Основні атрибути сутності: ідентифікатор, ідентифікатор користувача, кількість разів оцінювання від 1 до 5 балів.

Виходячи с перелічених сутностей було побудовано схему бази даних (рис. 2.39).

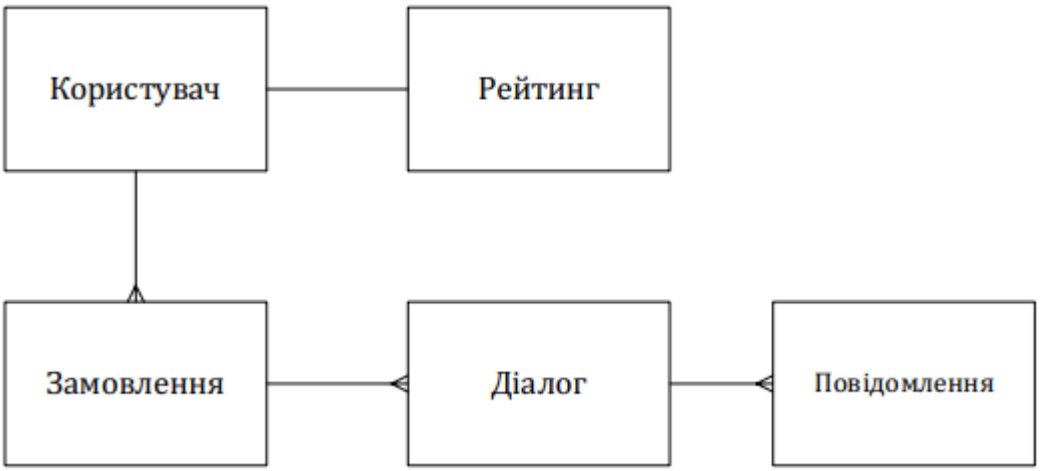


Рис. 2.39. Схема бази даних

### 2.4.3.2. Логічне проектування БД

База даних зберігатиметься на сервері Firebase, який має власну СКБД з NoSQL підходом [21]. Її структура передбачає гнучке формування сутностей: вони можуть бути вкладеними, а також кожне поле бази даних зберігається під ідентифікатором, який задається програмно або створюється автоматично. Також усі атрибути бази даних є дочірніми елементами ідентифікаторів, які в свою чергу вже зберігаються безпосередньо самими сутностями. СКБД не має можливості створення первинних та зовнішніх ключів, тому зв'язок між сутностями виконуються через програмне порівняння ідентифікаторів, а усі зв'язки на схемі бази даних відображаються умовно для зручності сприйняття.

Враховуючи дані особливості було проведено модифікацію бази даних, а саме: з сутностей були видалені ідентифікатори, так як їх значення знаходиться на рівень вище відносно атрибутів, додано додаткову сутність для прийняття виконавцями замовлень, сутність повідомлення вкладено у діалог, завдяки чому покращується компактність та зрозумілість бази даних: не потрібно зв'язувати дві сутності за допомогою первинних та зовнішніх ключів. При цьому усі атрибути повідомлення будуть у своєму початковому вигляді, але доступ до них відбуватиметься через ідентифікатор діалогу.

Відповідно до внесених змін було побудовано нову схему бази даних (рис. 2.40).


Рис. 2.40. Оновлена схема бази даних

Перелік основних сутностей бази даних:

– користувачі. Основні атрибути сутності: прізвище, ім’я, номер телефону, місце проживання, електронна адреса, тип користувача;

– замовлення. Основні атрибути сутності: ідентифікатор замовника, адреса, вартість, дата, час, тривалість, тип, статус, ідентифікатор виконавця, оцінка клієнта та виконавця;

– прийняття. Основні атрибути сутності: ідентифікатор виконавця, ідентифікатор замовлення;

– діалог. Основні атрибути сутності: ідентифікатор замовлення, ідентифікатор виконавця, список повідомлень (дата та час відправлення, тип відправника, текст повідомлення, статус читання);

– рейтинг. Основні атрибути сутності: ідентифікатор користувача, кількість разів оцінювання від 1 до 5 балів.

### 2.4.3.3. Фізичне проектування БД

Після логічного проектування бази даних було створено таблиці з переліком сутностей та атрибутів (табл. 2.17-2.22). Типи атрибутів не вказані, так як СКБД зберігає усю інформацію в однаковому форматі, яка в результаті може буде конвертована в будь-який тип.

Таблиця 2.17

**Сутність Користувач (User)**

| Заголовок | Ім’я поля | Додатковий опис |
|---|---|---|
| Прізвище | surname | Обов’язкове поле |
| Ім’я | name | Обов’язкове поле |
| Номер телефону | phone | Обов’язкове поле |
| Місто | city | Обов’язкове поле |
| Електронна адреса | mail | Додатково зберігається у розділі аутентифікації |
| Тип користувача | typeUser | Має два типи значень: замовник (client), виконавець (photographer) |

Таблиця 2.18

**Сутність Замовлення (Order)**

| Заголовок | Ім’я поля | Додатковий опис |
|---|---|---|
| Ідентифікатор замовника | idClient | Записується автоматично |
| Ідентифікатор виконавця | idPhotographer | Може бути порожнім, якщо замовник не обрав виконавця |
| Адреса | adress | Обов’язкове поле |
| Місто | city | Виділено окремо |
| Вартість | cost | Обов’язкове поле |
| Дата | date | Обов’язкове поле |
| Час | time | Обов’язкове поле |
| Тип | type | Обов’язкове поле |
| Статус | status | Має два значення: робоче та виконане |
| Оцінка клієнта | rateClient | Від1до5 |
| Оцінка виконавця | RatePhotographer | Від1до5 |

Таблиця 2.19

**Сутність Прийняття (Adoption)**

| Заголовок | Ім’я поля | Додатковий опис |
|---|---|---|
| Ідентифікатор виконавця | idPhotographer | Обов’язкове поле |
| Ідентифікатор замовлення | idOrder | Обов’язкове поле |

Таблиця 2.20

**Сутність Рейтинг (Raiting)**

| Заголовок | Ім’я поля | Додатковий опис |
|---|---|---|
| Ідентифікатор користувача | idUser | Обов’язкове поле |
| Кількість разів оцінювання в 1 бал | count1Start | За замовчанням 0 |
| Кількість разів оцінювання в 2 бали | count2Start | За замовчанням 0 |
| Кількість разів оцінювання в 3 бали | count3Start | За замовчанням 0 |
| Кількість разів оцінювання в 4 бали | count4Start | За замовчанням 0 |
| Кількість разів оцінювання в 5 бал | count5Start | За замовчанням 0 |

Таблиця 2.21

**Сутність Діалог (Dialog)**

| Заголовок | Ім’я поля | Додатковий опис |
|---|---|---|
| Ідентифікатор замовлення | idOrder | Обов’язкове поле |
| Ідентифікатор виконавця | idPhotographer | Обов’язкове поле |
| Повідомлення | Message | Дочірня сутність |

Таблиця 2.22

**Сутність Повідомлення (Message)**

| Заголовок | Ім’я поля | Додатковий опис |
|---|---|---|
| Дата | date | Визначається пристроєм |
| Час | time | Визначається пристроєм |
| Текст | text | Обов’язкове поле |
| Відправник | sender | Ідентифікатор користувача |
| Статус читання | read | За замовчанням false |

Було створено ER-діаграму, яка відображає сутності та їх атрибути (рис. 2.41). Так як у базі даних відсутні зв’язки, кожна сутність існує окремо від інших та не має зв’язків. Виключення становить лише Dialog, яка містить у собі дочірню сутність Message.

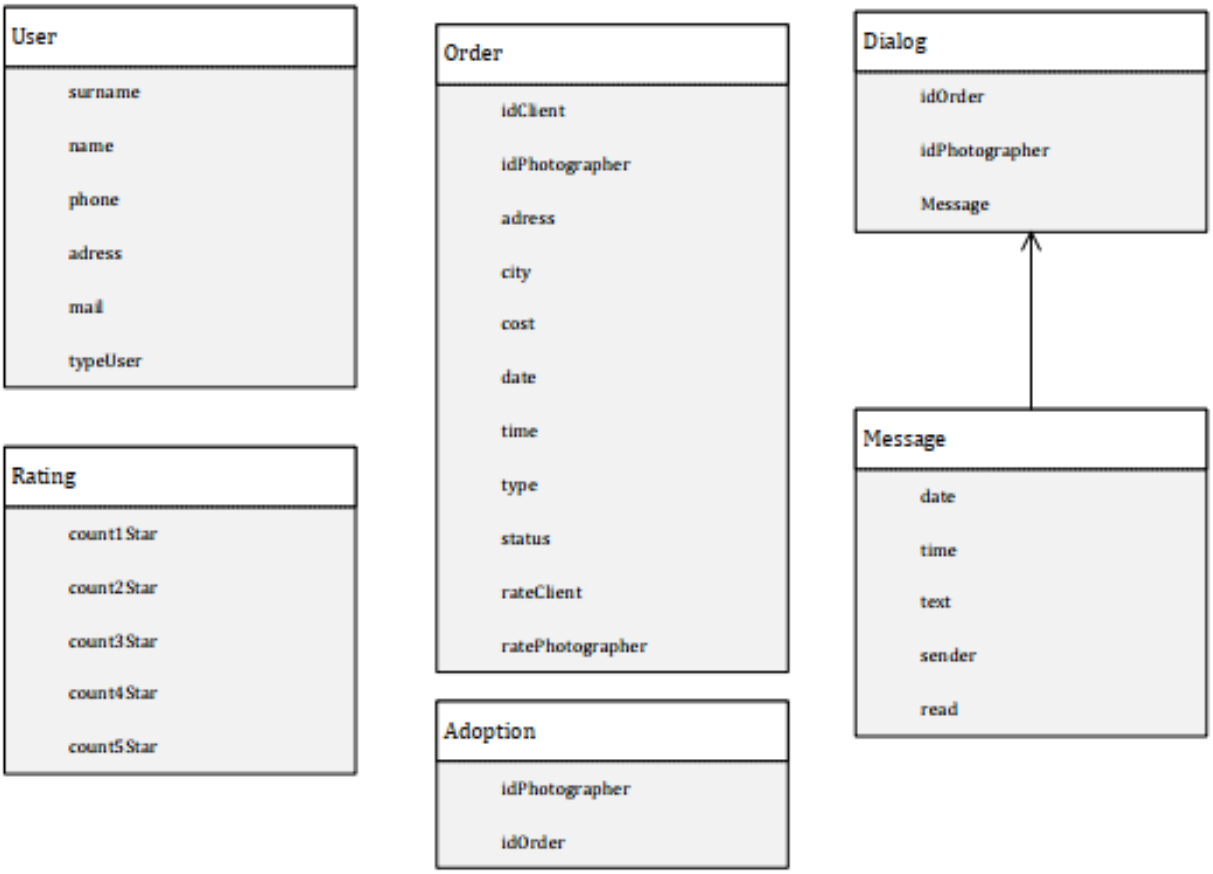


Рис. 2.41. ER-діаграма бази даних

### 2.4.3.4. Нормалізація бази даних

За рахунок специфіки структури СКБД, а також самої бази даних, необхідність в нормалізації відсутня, оскільки розбиття однієї сутності або атрибуту на декілька не покращить продуктивність роботи та не полегшить кодування програми. Доступ до даних виконується з використанням ключових слів, ідентифікаторів, а також користувацьких класів за допомогою яких можна створювати об’єкти та присвоювати значення атрибутів у конструкторі.

За рахунок гнучкості бази даних додання нових сутностей та атрибутів або редагування існуючих виконується без руйнування чи порушення початкової структури. Одна сутність може зберігати об’єкти з різною кількістю та назвами атрибутів, а тому раніше введена інформація зберігатиме власний вигляд в незалежності від нової. Це також породжує і недолік: при корінній зміні атрибутів чи програмного коду для обробки бази даних, необхідно провести очищення від попередньої інформації або перемістити нові дані в окрему сутність для запобігання виникнення помилок при роботі користувача з програмою. Якщо атрибут та користувацький клас мають різну структуру, то при отриманні інформації з бази даних будуть визначені лише поля, які співпадають за назвою.

## 2.5. Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними системи є дані, вносимі користувачем додатку для своєї ідентифікації та відправлення замовлення на пошук спеціаліста. До цих даних належать дані про:

– користувачів, ім’я, номер телефону, місце проживання, електронна адреса, тип користувача;

– замовлення:адреса, вартість, дата, час, тривалість, тип;

– діалог: текст повідомлення;

– оцінювання виконавця: від 1 до 5 балів.

Вся інформація про користувачів зберігається на сервері, що містить систему авторизації, базу даних, а також хмарне сховище.

Вихідним даними є наступні дії системи:

– формування рейтингу виконавців;

– створення замовлень для користувачів, які зареєстровані під роллю клієнтів;

– пошук виконавця та прийняття замовлення.

## 2.6. Опис роботи розробленої системи

### 2.6.1. Використані технічні засоби

При тестуванні перевіряється виконання усіх необхідних функцій додатку: робота з базою даних, коректність введення інформації, відображення усіх необхідних компонентів, оновлення даних у реальному часі.

Для тестування використовується емуляція мобільного пристрою на ПК за допомогою середовища Android Studio. У якості моделі пристрою було обрано Pixel 2 XL API 27 з версією операційної системи Android 8.1 та роздільною здатністю екрану 1440 x 2280 [14].

Емулятор має усі можливі функції, що виконує звичайний мобільний пристрій, а саме: робота з програмними додатками, підключення до інтернету, можливість створення та завантаження файлів (фото, текстові документи та інше), зміна орієнтації екрану (вертикальна та горизонтальна), введення даних у текстові поля, робота з діалоговими вікнами вибору значень або зображень і т.д.

### 2.6.2. Використані програмні засоби

Для розробки мобільного додатку застосовано високорівневу об’єктно-орієнтовану мову програмування Java, засоби якої забезпечують зручне та

структуроване написання коду. У якості середовища було використано Android Studio, що має підтримку Java і надає можливість редагування зовнішнього інтерфейсу програми через графічний редактор. Збереження даних виконується на сервері FireBase, який має інструменти для реалізації авторизації користувачів, а також набір бібліотек для роботи з Android Studio, за допомогою яких відбувається зчитування, або додавання інформації.

### 2.6.3. Виклик та завантаження програми

Для завантаження та встановлення програмного додатку на мобільний пристрій використовується файл з розширенням “.apk”, який генерується під час збірки проекту. Цей файл представляє собою архів, який містить інформацію про XML розмітку, класи, конфігурацію, додаткові ресурси та метадані (рис. 2.42). Розмір архіву складає менше 5 Мб.

Встановлення додатку виконується автоматично при натисканні на файл з потрібного Android пристрою, після чого на робочій області з’явиться ярлик для відкриття програми. За необхідності програма може провести запит на дозвіл використання певних функцій, якщо це вимагає пристрій.

| Имя | Размер | Сжат | Тип | Изменён | CRC32 |
|---|---|---|---|---|---|
| .. | | | Папка с файлами | | |
| res | | | Папка с файлами | | |
| okhttp3 | | | Папка с файлами | | |
| META-INF | | | Папка с файлами | | |
| resources.arsc | 523 800 | 523 800 | Файл "ARSC" | | 0DE1952A |
| play-services-ta... | 76 | 51 | Файл "PROPERTIES" | | 77B27071 |
| play-services-fla... | 76 | 51 | Файл "PROPERTIES" | | D4C1C2F2 |
| play-services-ba... | 82 | 53 | Файл "PROPERTIES" | | 4649E37E |
| play-services-ba... | 74 | 50 | Файл "PROPERTIES" | | 7FF6D65A |
| firebase-storage... | 70 | 48 | Файл "PROPERTIES" | | AC569037 |
| firebase-databas... | 94 | 56 | Файл "PROPERTIES" | | C0932E66 |
| firebase-databas... | 72 | 46 | Файл "PROPERTIES" | | DC6110FF |
| firebase-comm... | 68 | 47 | Файл "PROPERTIES" | | D71DF402 |
| firebase-auth-in... | 80 | 53 | Файл "PROPERTIES" | | E446E2BD |
| firebase-auth.pr... | 64 | 45 | Файл "PROPERTIES" | | EFF32C3F |
| classes.dex | 6 398 132 | 2 919 030 | Файл "DEX" | | F9D860C6 |
| AndroidManifes... | 7 756 | 2 081 | Документ XML | | 79109831 |

Рис. 2.42. Структура архіву APK

### 2.6.4. Опис інтерфейсу користувача

При першому запуску програми відкривається вікно аутентифікації з двома порожніми текстовими полями (рис. 2.43).


Рис. 2.43.Стартове вікно програми

При спробі ввести дані, які не містяться на сервері з’являється відповідне текстове повідомлення (рис. 2.44).

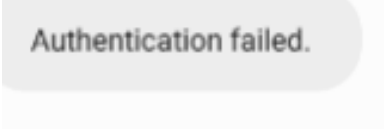


Рис. 2.44. Повідомлення про невдалу аутентифікацію

Для перевірки можливості створення нового профілю було проведено тестування реєстрації (рис. 2.45 – 2.47).

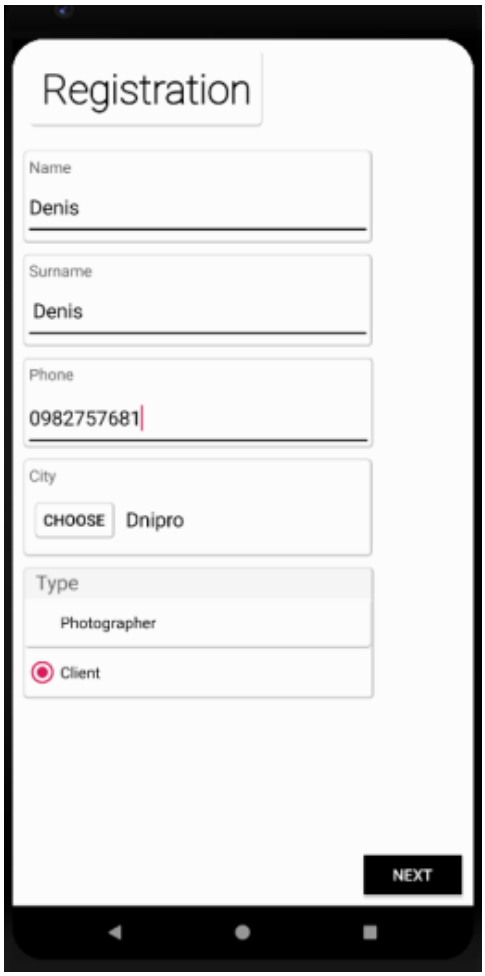


Рис. 2.45. Введення основних даних

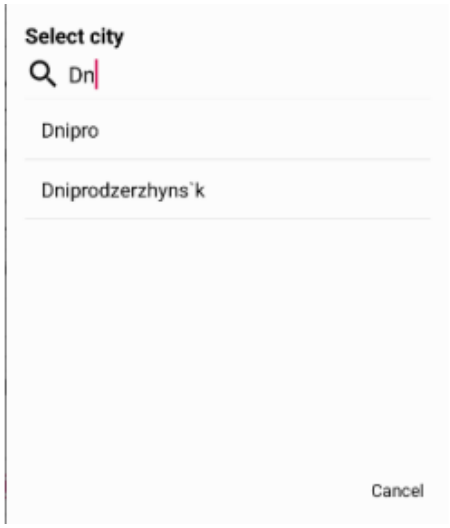


Рис. 2.46. Пошук міста у діалоговому вікні

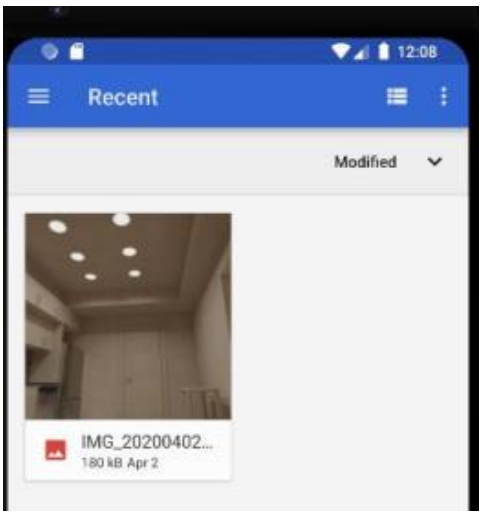


Рис. 2.47. Діалогове вікно вибору зображення

При успішній реєстрації з’являється повідомлення про виконання та відкривається вікно профілю з введеними даними, які також відображаються в шапці бічного меню (рис. 2.48 – 2.49).

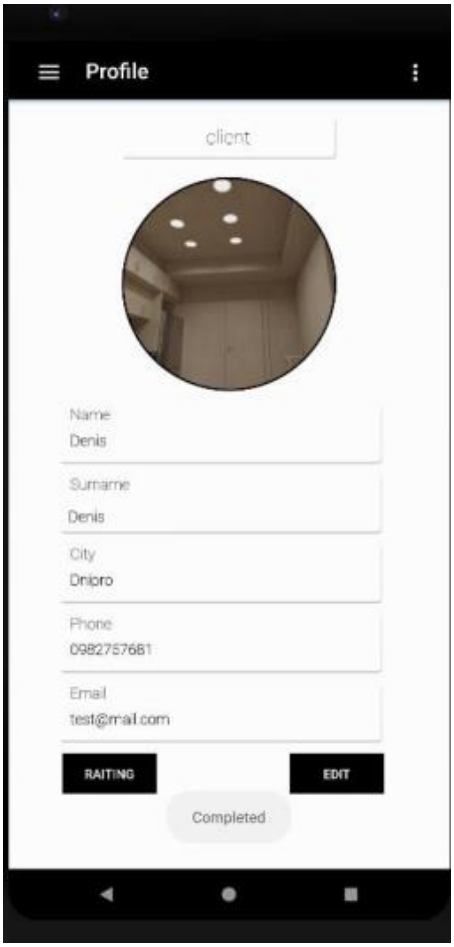


Рис. 2.48. Вікно профілю

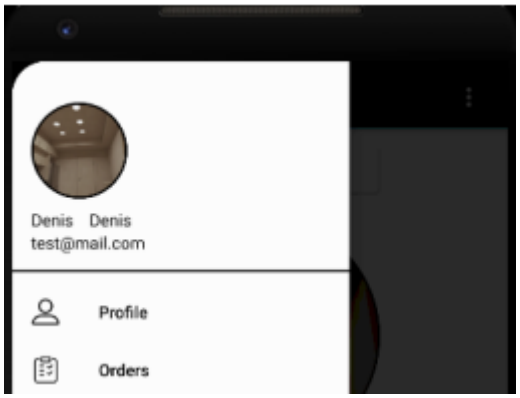


Рис. 2.49. Шапка бічного меню

При натисканні на кнопку редагування профілю з’являється вікно, у якому відображаються дані, що доступні для зміни (рис. 2.50).


Рис. 2.50. Вікно редагування профілю

При зміненні будь-яких даних та натисканні на кнопку збереження з’являється повідомлення про успішну зміну інформації (рис. 2.51).


Рис. 2.51. Повідомлення про зміну даних

Відповідні зміни відображаються також у вкладці профілю та на шапці.

При переході на вкладку замовлень з’являється нове вікно з порожнім список на кнопкою додання нового замовлення. При її натисканні з’являються форми для заповнення даних, в якому було введено інформацію замовлення з допомогою полів введення, діалогових вікон та випадаючого списку (рис. 2.52).


Рис. 2.52. Вікно додання замовлення

Після створення замовлення вікно закривається та з’являється відповідне повідомлення (рис. 2.53).

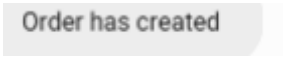


Рис. 2.53. Повідомлення про успішне додання

Для тестування відображення замовлень у списку було виконано ряд повторного додавання нових записів (рис. 2.54).

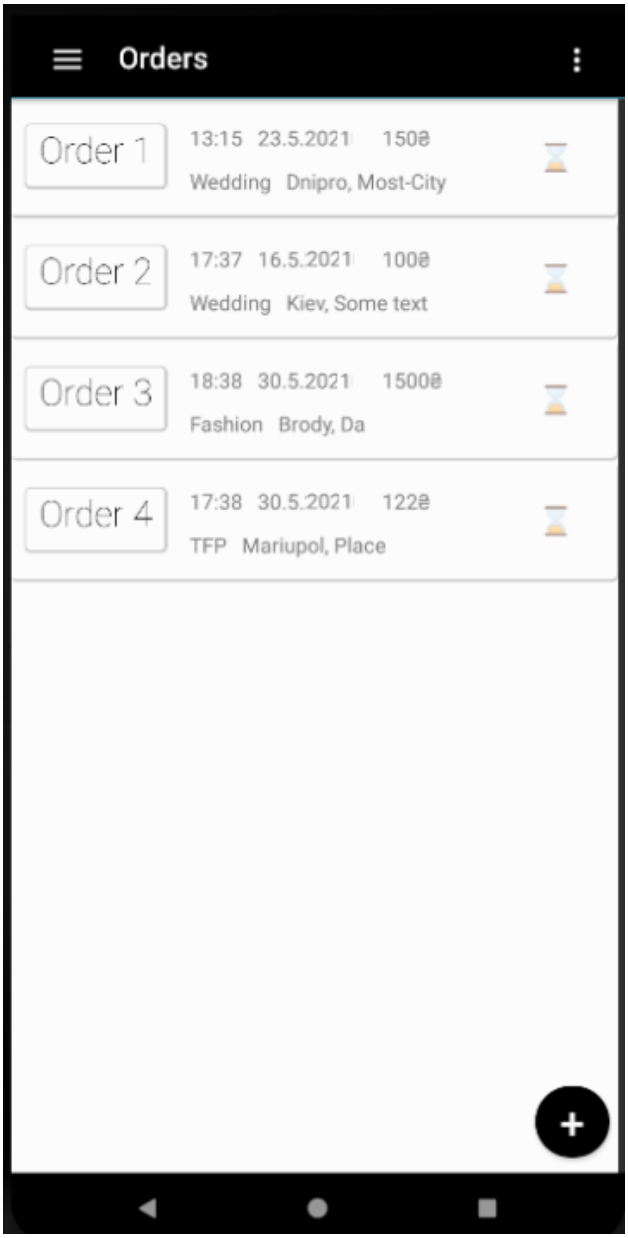


Рис.2.54. Список доданих замовлень

При натисканні на будь-яке замовлення відкривається нове вікно з детальною інформацією (рис. 2.55).


Рис. 2.55. Вікно детальної інформації

Редагування замовлення відбувається аналогічним чином, як і профілю: відкривається вікно, що містить заповнені поля введення інформації про замовлення. При зміні даних та натисканні на кнопку збереження вікно редагування закривається та з’являється відповідне повідомлення.

З метою перевірки перегляду даних було створено декілька додаткових профілів користувачів виконавців, які обрали дане замовлення. При натисканні

кнопки перегляду списку виконавців з’являється вікно зі створеними профілями (рис. 2.56).

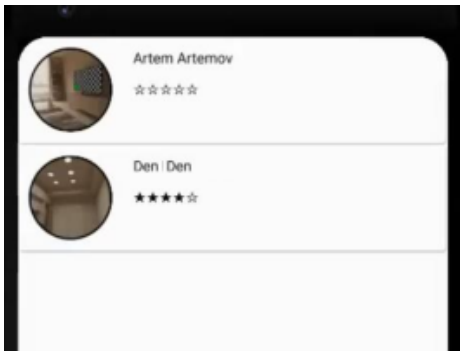


Рис. 2.56. Вікно списку виконавців

При натисканні на елемент списку з’являється детальна інформація про користувача (рис. 2.57).


Рис. 2.57. Вікно профілю обраного користувача

Якщо замовлення було виконане окрім основної інформації вікно профілю міститиме фрагмент для оцінювання роботи протилежної сторони (рис. 2.58). Оцінювання виконується лише один раз, після чого відповідна оцінка відображатиметься на тому ж самому місці (рис. 2.59).

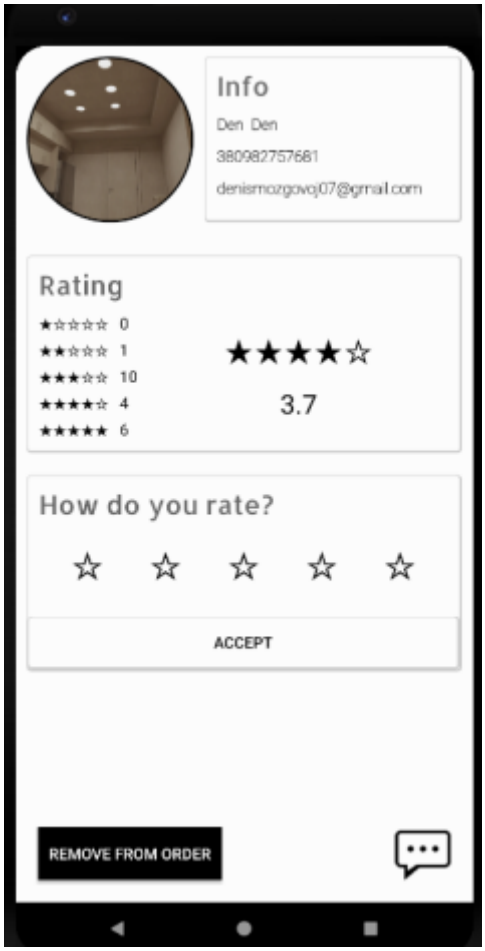


Рис. 2.58. Оцінювання протилежної сторони


Рис. 2.59. Значення поставленої оцінки

При натисканні на кнопку відкриття діалогу з’являється чат з введеними повідомленнями, а також ім’ям співрозмовника (рис. 2.60).

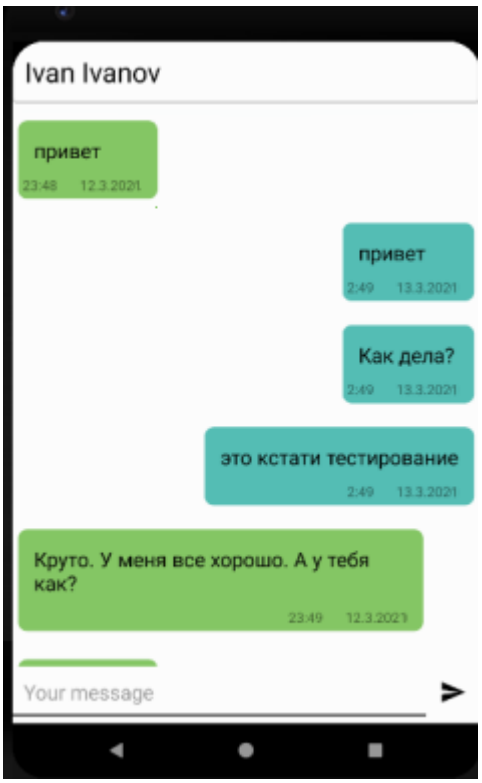


Рис. 2.60. Вікно діалогу

При написанні одного або декількох повідомлень відповідна інформація відображається на замовлення у отримувача, доки не буде відкритий діалог.

При авторизації зі сторони виконавця змінюється набір кнопок у бічному меню (рис. 2.61). З’являються такі пункти як прийняті та вільні замовлення. При переході на вкладку з вільними замовленнями з’являється список з можливістю фільтрації (рис. 2.62).

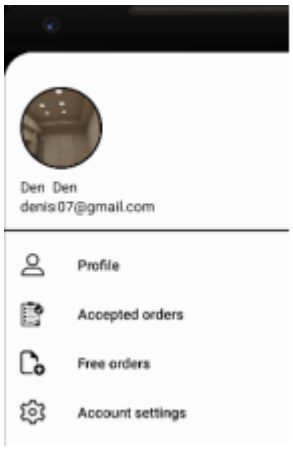


Рис. 2.61. Бічне меню виконавця

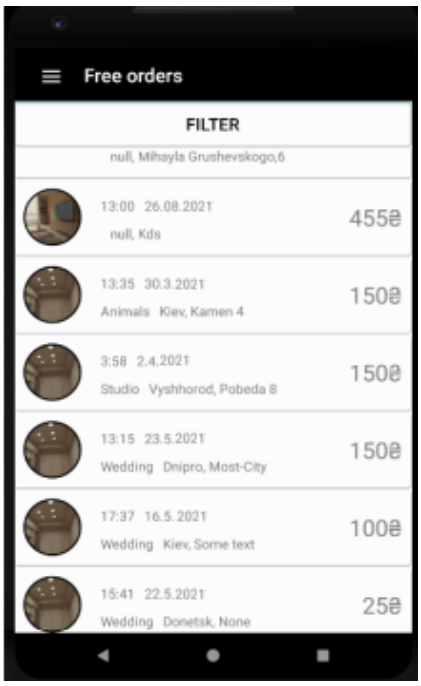


Рис. 2.62. Список вільних замовлень

При застосуванні фільтру з будь-яким набором значень (рис. 2.63) залишається лише частина замовлень, що відповідає умові фільтрації (рис. 2.64). Заповнення усіх полів фільтру не є обов'язковим. Пусте поле не буде включено в умову.


Рис. 2.63. Приклад фільтру

Якщо жодне з замовлень не відповідає умові фільтрації – список залишиться порожнім.

При повторному відкритті фільтру буде відображено попередньо введені значення. Якщо натиснути на кнопку “Clear” усі значення будуть очищенні, але дія фільтру зберігатиметься доки користувач не натисне кнопку прийняття змін “Accept”. При переході на інші вкладки бічного та поверненні назад інформація з фільтру буде очищена.

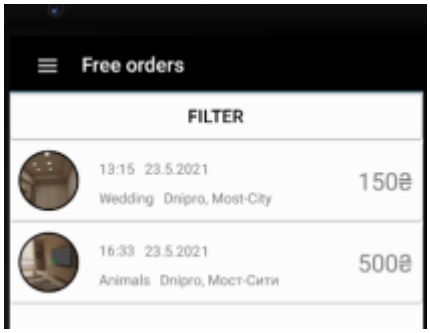


Рис. 2.64. Результат застосування фільтру

При натисканні на замовлення відкривається вікно з детальною інформацією та можливістю прийняття (рис. 2.65). Якщо прийняти замовлення – з’явиться відповідне повідомлення (рис. 2.66). При повторному натисканні буде виконано зворотню дію.


Рис. 2.65. Вікно вільного замовлення

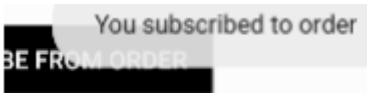


Рис. 2.66. Повідомлення про підписання

Після прийняття замовлення воно буде переміщено з вкладки вільних до вкладки прийнятих, в якій відображається аналогічний список за винятком того, що додатково кожний елемент містить статус (рис. 2.67).

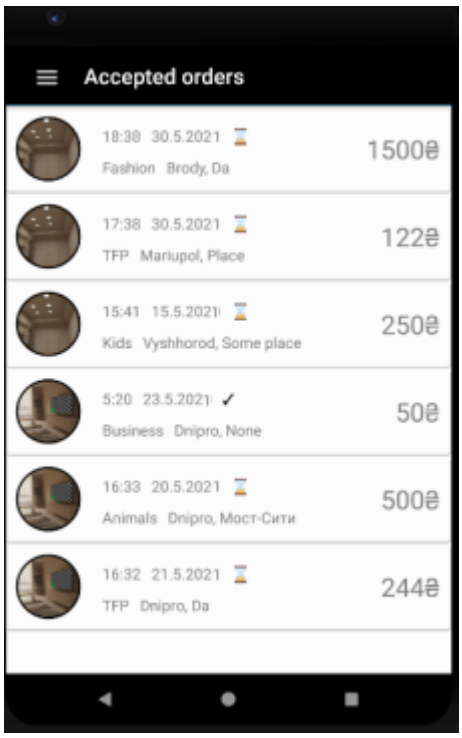


Рис. 2.67. Прийняті замовлення

Редагування даних аккаунту на відповідний вкладці супроводжується повідомленням про успішну або невдало проведену операцію (рис. 2.68-2.76).


Рис. 2.68. Вікно налаштувань аккаунту


Рис. 2.69. Повідомлення про зміну паролю


Рис. 2.70. Повідомлення про зміну пошти

# РОЗДІЛ 3
# ЕКОНОМІЧНИЙ РОЗДІЛ

## 3.1.Розрахунок трудомісткості та вартості розробки програмного продукту

Вихідні дані трудомісткості розробки програмного забезпечення:

- передбачувана кількість операторів – 1300;
- коефіцієнт складності програми – 1;
- коефіцієнт корекції програми в ході її розробки - 0,5;
- годинна заробітна плата програміста, грн/год – 30;
- вартість машино-години ЕОМ, грн/год – 5.

Розрахунок показників трудомісткості при створенні програмного забезпечення ускладнюється в силу неоднозначності витрачається часу для різних ділянок коду програми. Трудомісткість розробки програмного забезпечення прийнято розраховувати на підставі моделей з різною точністю оцінки.

$$t = t0 + ti + t\,a + tп + tотл + tд\text{ , люд. год.,} \qquad (3.1)$$

де t0 – затрати праці на підготовку і опис поставленої задачі (приймається = 45), чол-год;

ti – витрати праці на дослідження розв'язку задачі, чол-год;

t a – витрати праці на розробку блок–схеми алгоритму, чол-год;

tп – витрати праці на програмування по готовій блок–схемі, чол-год;

tотл – витрати праці на відладку, чол-год;

tд – витрати праці на підготовку документації по завданню, люд. год.

Складові витрат праці визначаються виходячи з умовного числа операторів, які необхідно написати в процесі роботи над програмою з урахуванням можливих уточнень в постановці завдання і вдосконалення алгоритму.

Умовне число операторів у програмі:

$$Q = q \cdot c(1 + p), \tag{3.2}$$

де q – передбачувана кількість операторів;

с – коефіцієнт складності програми;

р – коефіцієнт корекції програми в ході її розробки.

$$Q = 1300 \cdot 1 \cdot (1 + 0{,}5) = 1950. \tag{3.3}$$

Витрати праці на вивчення опису завдання ti визначаються з урахуванням уточнення опису і кваліфікації програміста за формулою:

$$t_u = \frac{Q \cdot B}{(75...85)K}, \text{ чол-год}, \tag{3.4}$$

де В – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі, В=1.2 ... 1.5;

К – коефіцієнт кваліфікації програміста, який визначається в залежності від стажу роботи за даною спеціальністю. Він становить при стажі роботи, років: до 2 – 0.8;

$$t_u = \frac{1950 \cdot 1{,}2}{85 \cdot 0{,}8} = 36.56, \text{люд. год.} \tag{3.5}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25)K}; \tag{3.6}$$

$$t_a = \frac{1950}{20 \cdot 0{,}8} = 121.87, \text{ люд. год.} \tag{3.7}$$

Витрати праці на складання програми по готовій блок–схемі розраховуються за формулою:

$$t_n = \frac{Q}{(20...25)K}; \tag{3.8}$$

$$t_n = \frac{1950}{25 \cdot 0{,}8} = 97.5, \text{ люд. год.} \tag{3.9}$$

Витрати праці на відладку програми на ЕОМ, розраховується за формулою з умовою автономної налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5)K}; \tag{3.10}$$

$$t_{îòë} = \frac{1950}{5 \cdot 0{,}8} = 812.5, \text{ люд. год.,} \tag{3.11}$$

Витрати праці на підготовку документації по завданню визначаються за формулою:

$$t_д = t_{др} + t_{до}; \tag{3.12}$$

де tдр – трудомісткість підготовки матеріалів до рукопису;

tдо – трудомісткість редагування, друку та оформлення документації.

$$t_{др} = \frac{Q}{(15...20)K}; \tag{3.13}$$

$$t_{äð} = \frac{1950}{20 \cdot 0{,}8} = 162.5, \text{ люд. год.} \tag{3.14}$$

$$t_{до} = 0{,}75 \cdot t_{др}; \tag{3.15}$$

$$t_{äî} = 0{,}75 \cdot 162.5 = 121.87, \text{ люд. год.} \tag{3.16}$$

$$t_ä = 162.5 + 121.87 = 284.37, \text{ люд. год.} \tag{3.17}$$

Отримуємо трудомісткість розробки програмного забезпечення:

$$t = 45 + 36{,}56 + 121{,}87 + 97{,}5 + 812{,}5 + 284{,}37 = 1382{,}81 \text{ люд. год.} \quad (3.18)$$

## 3.2. Розрахунок витрат на створення програмного забезпечення

Витрати на створення програмного забезпечення (Кпо) складаються з витрат на заробітну плату розробників програми (Ззп), яка визначається множенням сумарної трудомісткості розробки ПО (t) на середню заробітну плату програміста з нарахуваннями, а також вартості машинного часу на налагодження програми на ЕОМ (Змв), вона визначається виходячи з вартості 1-го години машинного часу, конкретного типу ЕОМ і витрат машинного часу на налагодження.

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \quad (3.19)$$

де Ззп – заробітна плата розробників визначається за формулою:

$З_{ЗП} = t \cdot C_{ПР}$, грн,

де t – загальна трудомісткість розробки ПО;

Спр – середня годинна заробітна плата програміста.

$$С_{ЗП} = 1382 \cdot 30 = 41550, \text{грн.} \quad (3.20)$$

Змв – вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot C_{М}, \text{ грн,} \quad (3.21)$$

де tотл – трудомісткість налагодження програми на ЕОМ;

СМЧ – вартість машино-години ЕОМ.

$$Ç_{ÌÂ} = 812 \cdot 5 = 4062, \text{ грн.} \quad (3.22)$$

$$\hat{E}_{ïî} = 41550 + 4062 = 45612 \text{ , грн.} \quad (3.33)$$

Очікувана тривалість розробки:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.} \quad (3.34)$$

де Bk - число розробників;

Fp - місячний фонд робочого часу (при 40 годинному робочому тижні Fp=176 годин).

$$T = \frac{1382}{1*176} = 7.8 \text{ міс.} \quad (3.35)$$

У результаті з'ясувалося, що в загальній складності необхідно 1382,8 люд. год. для розробки даного програмного забезпечення та 45612 грн. на її розробку, очікувана тривалість розробки складе 7,8 місяця.

# ВИСНОВКИ

Враховуючи сучасні потреби взаємодії клієнтів та виконавців у галузі надання послуг з фото та відеозйомки, було створено мобільний додаток freelance-систему PhotoFree.

Для розробки мобільного додатку застосовано високорівневу об’єктно-орієнтовану мову програмування Java, засоби якої забезпечують зручне та структуроване написання коду. У якості середовища було використано Android Studio, що має підтримку Java і надає можливість редагування зовнішнього інтерфейсу програми через графічний редактор. Збереження даних виконується на сервері FireBase, який має інструменти для реалізації авторизації користувачів, а також набір бібліотек для роботи з Android Studio, за допомогою яких відбувається зчитування, або додавання інформації.

В постановці завдання описано функціонал, який повинен забезпечувати мобільний додаток: реєстрація, авторизація, робота з власними персональними даними, збереження даних на сервері, меню навігації, робота з замовленнями та перегляд рейтингу користувачів. Визначено, що для оптимальної роботи слід використовувати операційну систему Android, починаючи з версії 8.0.

На етапі проектування були створені макети зовнішнього інтерфейсу програми, що складаються з візуальних та невізуальних компонентів. Враховуючи те, що програма призначена для користувачів двох типів, що мають різний набір функціоналу, при проектуванні певних вікон або компонентів за основу використовувалися раніше створені елементи, що значно прискорювало розробку. Також з урахуванням особливостей СКБД було спроектовано базу даних. Обробка даних виконується безпосередньо на мобільному пристрої, відображення оновленої інформації виконується у реальному часі без необхідності перезавантаження сторінок. При розробці програмного додатку були використані користувацькі класи.

В результаті отримано файл з розширенням “.apk”, за допомогою якого виконується встановлення додатку на пристрій. Тестування було виконано за

допомогою емулятора мобільного пристрою з операційною системою Android, наданий середовищем розробки. Перевірено коректність відображення усіх візуальних компонентів, обробку виключних ситуацій, а також непередбачених дій користувача.

Завдяки розробленому додатку всім користувачам надається можливість легко та швидко отримувати актуальну інформацію щодо надання послуг спеціалістів з фото- та відеозйомки, створювати замовлення, переглядати історії замовлень та редагувати персональні дані у системі.

Розроблене програмне забезпечення дозволить створити зручну мобільну структуровану систему для вирішення проблеми пошуку спеціалістів для виконання необхідного завдання у таких сферах діяльності, як фотографія та відеозйомка.

В економічному розділі з'ясувалося, що в загальній складності необхідно 1382,8 люд. год. для розробки даного програмного забезпечення та 45612 грн. на її розробку., очікувана тривалість розробки складе 7,8 місяця.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання : (ГОСТ 7.1-2003, IDT) : ДСТУ ГОСТ 7.1:2006. – Чинний з 2007–07–01. – К. : Держспоживстандарт України, 2007. – 47 с. – (Система стандартів з інформації, бібліотечної та видавничої справи) (Національний стандарт України).

2. Бусигін Б.С., Коротенко Г.М., Коротенко Л.М. Прикладна інформатика. Підручник для студентів комп'ютерних спеціальностей. – Дніпропетровськ: Видавництво НГУ, 2004. – 559 с. URL: http://www.programmer.dp.ua/book-ua-k01.php. дата звернення: 22.04.2021.

3. Бусыгин Б.С., Дивизинюк М.М., Коротенко Г.М., Коротенко Л.М. Введение в современную информатику. Учебник. – Севастополь: Издательство СНУЯЭиП, 2005. – 644 с. / URL: http://www.programmer.dp.ua/book-ru-k02.php. дата звернення: 15.01.2018.

4. Дейтел Х. М. Android для разработчиков / Х. М. Дейтел, П. Д. Дейтел. – Санкт-Петербург: Питер Пресс, 2015. – 384 с.

5. Державне управління фрілансовою діяльністю в умовах розвитку національного ринку: монографія / О. Є. Кузьмін, О. Г. Мельник, О. С. Скибінський, Л. О. Саталкіна, Н. Ю. Реверенда; Нац. ун-т «Львів. політехніка». - Львів: Центр Європи, 2016. - 166 с. - Бібліогр.: с. 141-160.

6. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення : ДСТУ 3008-95. – Чинний від 1996–01–01. – К. : Держстандарт України, 1996. – 39 с.

7. ДСТУ 2394-94 Інформація та документація. Комплектування фонду, бібліографічний опис, аналіз документів. Терміни та визначення. – Чинний від 01.01.1995. - Київ : Держстандарт України, 1994. – 88 с.

8. Кожомбердиева Г.И. Конспект Java – Гродно: ГрГУ Я.Купалы, 2010. – 37 с.– (кн. 1).

9. Інтеграція FireBase в Android . – 2019. –URL: https://codelabs. developers. google.com/ codelabs/firebase-android-ru/index.html?index=..%2F..lang-ru#0. дата звернення: 22.04.2021.

10. Леслі Д. Разработка приложений для Android-устройств. Т. 1: Базовые принципы / Дерсі Леслі. – Москва: Лорі, 2014. – 402 с. –(5).

11. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності “Комп’ютерні системи ” / Укладачі О.Г. Вагонова, Нікітіна О.Б. Н.Н. Романюк – Дніпропетровськ: Національний гірничий університет. – 2013. – 23с.

12. Методичні рекомендації до виконання кваліфікаційних робіт бакалаврів напряму підготовки 122 «Комп’ютерні науки» галузі знань 12 Інформаційні технології/, Л.М. Коротенко , О.С. Шевцова; Нац. гірн. ун-т. – Д : ДВНЗ НГУ, 2019. – 65 с.

13. Методичні рекомендації щодо написання, оформлення та представлення учнівських науково-дослідницьких робіт учнів – членів Малої академії наук України / Г.Г. Півняк, Л.М. Коротенко, І.М. Удовик, Є.М. Головня – Д.: ДВНЗ «Національний гірничий університет», 2017. – 24 с.

14. Настройка Android-эмулятора на Android // o7planning – URL: https://o7planning.org/ru/10413/configuring-android-emulator-android -studio. дата звернення: 22.04.2021.

15. Скорочення слів в українській мові у бібліографічному описі. Загальні правила та вимоги : ДСТУ 3582-97. – Чинний від 1998–07–01. – К. : Держстандарт України, 1998. – 24 с. – (Державний стандарт України).

16. Структура проекта в Android Studio // JavaDevBlog. – 2016. – URL: https://javadevblog.com/struktura-proekta-v-android-studio.html. дата звернення: 22.04.2021.

17. Фрилансер / URL: https://uk.wikipedia.org/wiki/Фрилансер/ дата звернення: 22.04.2021.

18. Що таке мова XML? // 1. – 2019. – URL: https://serpstat.com/ru /blog/chto-takoe-jazyk-xml/. дата звернення: 22.04.2021

19. Ян К. Проектирование пользовательского интерфейса в Android / Кліфтон Ян. – Москва: ДМК-Пресс, 2017. – 452 с.

20. Kumar A. Освоєння Firebase для розробки під Android / Ashok Kumar. – Нью-Делі: Packt Publishing, 2018. – 394 с. – (1). – (1; 1).

21. Firebase// Wikipedia. – 2019. – URL: https://ru.wikipedia.org/wiki/ Firebase. дата звернення: 22.04.2021.

**ДОДАТОК А**

# КОД ПРОГРАМИ

**DatabaseHelper:**

```
package com.example.dyplom.Custom_classes;
public class DatabaseHelper {
private FirebaseDatabase database;
private DatabaseReference dbRef;
private List<Order> orders = new ArrayList<>(); private List<User> photgraphers = new
ArrayList<>(); private List<Message> messages = new ArrayList<>(); private List<Order>
freeOrders = new ArrayList<>(); private List<Order> notAcceptedOrders = new ArrayList<>();
private List<Order> acceptedOrders = new ArrayList<>();
public interface DataStatusUser{
void DataIsLoaded(User user);
void DataIsInserted();
void DataIsUpdated();
void DataIsDeleted();
}
public interface DataStatusListPhotographers{
void DataIsLoaded(List<User> photographers);
}
public interface PhtID{
void DataIsLoaded(List<String> photographers);
}
public interface OrdersID{
void DataIsLoaded(List<String> orders);
}
public interface CountMsg
{
void DataIsLoaded(int count);
}
public interface DataStatusRating{
void DataIsLoaded(Rating rating, String idRating);
}
public interface DataStatusOrder{
void DataIsLoaded(Order          order);
void ListIsLoaded(List<Order> orders);
void DataIsInserted();
void DataIsUpdated();
void DataIsDeleted();
}
public interface DataStatusDialog{
void ListIsLoaded(List<Message> messages, String roomId); void MessageIsAdded();
}
public          DatabaseHelper()
{
database = FirebaseDatabase.getInstance();
dbRef = database.getReference();
}
```

```
public void GetUser(final String userId , final DataStatusUser dataStatus)
{
dbRef.child("Users").child(userId).addValueEventListener(new ValueEventListener() { @Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) { User user =
dataSnapshot.getValue(User.class);
user.setId(userId);
dataStatus.DataIsLoaded(user);
}
@Override
public void onCancelled(@NonNull DatabaseError databaseError) {
}
});
}
public void CreateUser(String userId, User user, final DataStatusUser dataStatus)
{
dbRef.child("Users").child(userId).setValue(user, new DatabaseReference.CompletionListener() {
@Override
public void onComplete(@Nullable DatabaseError databaseError, @NonNull DatabaseReference
databaseReference) {
dataStatus.DataIsInserted();
}
});
}
public void UpdateUser(String userId, final User changedUser , final DataStatusUser dataStatus) {
dbRef.child("Users").child(userId).setValue(changedUser, new
DatabaseReference.CompletionListener() { @Override
public void onComplete(@Nullable DatabaseError databaseError, @NonNull DatabaseReference
databaseReference) {
dataStatus.DataIsUpdated();
dataStatus.DataIsLoaded(changedUser);
}
});
}
public void AddOrder(Order order, final DataStatusOrder dataStatus)
{
dbRef.child("Orders").push().setValue(order, new DatabaseReference.CompletionListener() {
@Override
public void onComplete(@Nullable DatabaseError databaseError, @NonNull DatabaseReference
databaseReference) {
dataStatus.DataIsInserted();
}
});
}
public void GetOrder(final String orderId, final DataStatusOrder dataStatus)
{
dbRef.child("Orders").child(orderId).addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) { Order order;
order = dataSnapshot.getValue(Order.class);
if(order!=null)
{
order.SetId(orderId);
```

```
dataStatus.DataIsLoaded(order);
}
}
@Override
public void onCancelled(@NonNull DatabaseError databaseError) {
}
});
}
public void DeleteOrder(final String orderId)
{
dbRef.child("Orders").child(orderId).removeValue();
}
public void UpdateOrder(final String orderId, Order order)
{
dbRef.child("Orders").child(orderId).setValue(order);
}
public void ChangeStatusOrder(final String orderId, final String status)
{
dbRef.child("Orders").child(orderId).child("orderStatus").setValue(status);
}
public void ReadOrdersForUser(final String userID, final DataStatusOrder dataStatus)
{
dbRef.child("Orders").addValueEventListener(new ValueEventListener() { @Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) { orders.clear();
for(DataSnapshot keyNode : dataSnapshot.getChildren())
{
Order order = keyNode.getValue(Order.class);
order.SetId(keyNode.getKey());
if(order.getIdClient().equals(userID))
{
orders.add(order);
}
}
dataStatus.ListIsLoaded(orders);
}
});
}
public void GetPhotographersId(final String IdOrder, final PhtID dataStatus)
{
final List<String> phtId = new ArrayList<>();

dbRef.child("Adoption").addValueEventListener(new ValueEventListener() { @Override

public void onDataChange(@NonNull DataSnapshot dataSnapshot) { for(DataSnapshot keyNode :
dataSnapshot.getChildren()) {

String currentIdOrder = keyNode.child("idOrder").getValue().toString(); if
(currentIdOrder.equals(IdOrder)) {
phtId.add(keyNode.child("idPhotographer").getValue().toString());
}
}
dataStatus.DataIsLoaded(phtId);
```

```
}
});
}
public void ReadPhotographers(final List<String> id, final DataStatusListPhotographers dataStatus)
{
dbRef.child("Users").addValueEventListener(new ValueEventListener() { @Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) { for(DataSnapshot keyNode :
dataSnapshot.getChildren()) {
for(int i = 0; i<id.size();i++)
{
if(id.get(i).equals(keyNode.getKey()))
{
User user = keyNode.getValue(User.class);
user.setId(id.get(i));
photgraphers.add(user);
}
}
}
dataStatus.DataIsLoaded(photgraphers);
}
@Override
public void onCancelled(@NonNull DatabaseError databaseError) {
}
});
}
public void GetRating(final        String userId, final DataStatusRating dataStatusRating)
{
dbRef.child("Rating").addValueEventListener(new ValueEventListener() { @Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) { for(DataSnapshot keyNode :
dataSnapshot.getChildren()) {
if(keyNode.child("idUser").getValue().toString().equals(userId))
{
Rating rating = keyNode.getValue(Rating.class);
dataStatusRating.DataIsLoaded(rating, keyNode.getKey());
}
}
}
});
}
public void SetRaiting(final String idRating, Rating rating)
{
if(idRating!=null)
dbRef.child("Rating").child(idRating).setValue(rating);
else
dbRef.child("Rating").push().setValue(rating);
}
public void SetRateClient(final String idOrder, final int value)
{
dbRef.child("Orders").child(idOrder).child("rateClient").setValue(value);
}
public void SetRatePhotographer(final String idOrder,    final int value)
{
```

```
dbRef.child("Orders").child(idOrder).child("ratePhotographer").setValue(value);
}
public void AcceptPhotographerToOrder(String idOrder, String idPhotographer)
{
dbRef.child("Orders").child(idOrder).child("idPhotographer").setValue(idPhotographer);
}
public void RemovePhotographerFromOrder(String idOrder)
{
dbRef.child("Orders").child(idOrder).child("idPhotographer").setValue("");
}
public void GetDialogRoom(final String idPhotographer, final String idOrder, final
DataStatusDialog dataStatus)
{
messages.clear();
dbRef.child("Dialogs").addValueEventListener(new ValueEventListener() { @Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) { for(DataSnapshot keyNode :
dataSnapshot.getChildren()) {
if(keyNode.child("idPhotographer").getValue().toString().equals(idPhotographer) &&
keyNode.child("idOrder").getValue().toString().equals(idOrder))
{
final String idRoom = keyNode.getKey();
Log.d("Info", "Entered to room");

dbRef.child("Dialogs").child(keyNode.getKey()).child("Messages").addValueEventListener(new
ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) { messages.clear();
for(DataSnapshot keyNode : dataSnapshot.getChildren())
{
Log.d("Info", "Entered to messages");
Message message = keyNode.getValue(Message.class);
messages.add(message);
}
dataStatus.ListIsLoaded(messages, idRoom);
}
@Override
public void onCancelled(@NonNull DatabaseError databaseError) {
}
});
return;
}
}
dbRef.child("Dialogs").push().setValue(new Dialog(idOrder, idPhotographer), new
DatabaseReference.CompletionListener() {
@Override
public void onComplete(@Nullable DatabaseError databaseError, @NonNull DatabaseReference
databaseReference) {
Log.d("Info",databaseReference.getKey()); dataStatus.ListIsLoaded(messages,
databaseReference.getKey());
}
});
}
```

```
@Override
public void onCancelled(@NonNull DatabaseError databaseError) {
}
});
}
public void SendMessage(String idRoom, Message message)
{
dbRef.child("Dialogs").child(idRoom).child("Messages").push().setValue(message);
}
public void ReadFreeOrders(final String idPhotographer, final DataStatusOrder dataStatus)
{
dbRef.child("Orders").addValueEventListener(new ValueEventListener() { @Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) { freeOrders.clear();
for(DataSnapshot keyNode : dataSnapshot.getChildren())
{
Log.d("INFO", "Entered to orders");
final Order order = keyNode.getValue(Order.class);
order.SetId(keyNode.getKey());
if(order.getIdPhotographer() == null || order.getIdPhotographer().isEmpty())
freeOrders.add(order);
}
dataStatus.ListIsLoaded(freeOrders);
}
});
}
public void ReadNotAcceptedOrders(final List<Order> orders, final String idPhotographer, final
DataStatusOrder dataStatus)
{
dbRef.child("Adoption").addValueEventListener(new ValueEventListener() { @Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
notAcceptedOrders.clear();
for(int i = 0; i <orders.size(); i++) {
boolean isAccepted = false;
for (DataSnapshot keyNode : dataSnapshot.getChildren()) {
if (orders.get(i).getId().equals(keyNode.child("idOrder").getValue().toString())
&&
idPhotographer.equals(keyNode.child("idPhotographer").getValue().toString())) { isAccepted =
true;
break;
}
}
if(!isAccepted) notAcceptedOrders.add(orders.get(i));
}
dataStatus.ListIsLoaded(notAcceptedOrders);
}
@Override
public void onCancelled(@NonNull DatabaseError databaseError) {
}
});
}
public void SubscribePhotographer(String idOrder, String idPhotographer)
{
```

```
dbRef.child("Adoption").push().setValue(new Adoption(idOrder,idPhotographer));
}
public void UnscribePhotographer(final String idOrder, final String idPhotographer)
{
dbRef.child("Adoption").addValueEventListener(new ValueEventListener() { @Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) { for (DataSnapshot keyNode :
dataSnapshot.getChildren()) {
if(idOrder.equals(keyNode.child("idOrder").getValue().toString()) &&
idPhotographer.equals(keyNode.child("idPhotographer").getValue().toString()))
{
dbRef.child("Adoption").child(keyNode.getKey()).removeValue();
}
}
}
@Override
public void onCancelled(@NonNull DatabaseError databaseError) {
}
});
}
public void GetIdAcceptedOrders(final String idPhotographer,final OrdersID dataStatus)
{
final List<String> ordersId = new ArrayList<>();
dbRef.child("Adoption").addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) { ordersId.clear();
for(DataSnapshot keyNode : dataSnapshot.getChildren()) {
if (idPhotographer.equals(keyNode.child("idPhotographer").getValue().toString())) {
ordersId.add(keyNode.child("idOrder").getValue().toString());
}
}
dataStatus.DataIsLoaded(ordersId);
}
});
}
public void GetAcceptedOrders(final String idPhotographer,final List<String> idOrders, final
DataStatusOrder dataStatus)
{
dbRef.child("Orders").addValueEventListener(new ValueEventListener() { @Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
acceptedOrders.clear();
for(DataSnapshot keyNode : dataSnapshot.getChildren())
{
for(int i = 0; i<idOrders.size();i++)
{
if(idOrders.get(i).equals(keyNode.getKey()))
{
Order order = keyNode.getValue(Order.class);
order.SetId(idOrders.get(i));
acceptedOrders.add(order);
}
}
}
```

```
dataStatus.ListIsLoaded(acceptedOrders);
}
@Override
public void onCancelled(@NonNull DatabaseError databaseError) {
}
});
}
public void GetCountOfUnreadMsg(final String idOrder,final String typeUser,final CountMsg
dataStatus) {
final List<Message> unreadMsg = new ArrayList<>();
dbRef.child("Dialogs").addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) { unreadMsg.clear();
for(DataSnapshot keyNode : dataSnapshot.getChildren())
{
if(keyNode.child("idOrder").getValue().toString().equals(idOrder))
{
final String idRoom = keyNode.getKey();
dbRef.child("Dialogs").child(keyNode.getKey()).child("Messages").addValueEventListener(new
ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
for(DataSnapshot keyNode : dataSnapshot.getChildren())
{
Message message = keyNode.getValue(Message.class); if(!message.isRead() &&
message.getSender().equals(typeUser))
unreadMsg.add(message);
}
dataStatus.DataIsLoaded(unreadMsg.size());
}
});
return;
}
}
}
});
}
public void GetIndCountOfUnreadMsg(final String idOrder, final String idPhotographer,final
String typeUser,final CountMsg dataStatus)
{
final List<Message> unreadMsg = new ArrayList<>();
dbRef.child("Dialogs").addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) { unreadMsg.clear();
for(DataSnapshot keyNode : dataSnapshot.getChildren())
{

if(keyNode.child("idOrder").getValue().toString().equals(idOrder) &&
keyNode.child("idPhotographer").getValue().equals(idPhotographer))
{
final String idRoom = keyNode.getKey();
```

```
dbRef.child("Dialogs").child(keyNode.getKey()).child("Messages").addValueEventListener(new
ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
for(DataSnapshot keyNode : dataSnapshot.getChildren())
{
Message message = keyNode.getValue(Message.class); if(!message.isRead() &&
message.getSender().equals(typeUser)) unreadMsg.add(message);
}
dataStatus.DataIsLoaded(unreadMsg.size());
}
@Override
public void onCancelled(@NonNull DatabaseError databaseError) {
}
});
return;
}
}
}
});
}
public void ReadMessages(final String idOrder ,final String idPhotographer, final String typeUser)
{
final Handler handler = new Handler();
handler.postDelayed(new Runnable() {
@Override
public void run() {
dbRef.child("Dialogs").addListenerForSingleValueEvent(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
for(DataSnapshot keyNode : dataSnapshot.getChildren())
{
final String key = keyNode.getKey();
if(keyNode.child("idOrder").getValue().toString().equals(idOrder) &&
keyNode.child("idPhotographer").getValue().equals(idPhotographer))
{
final String idRoom = keyNode.getKey();
dbRef.child("Dialogs").child(keyNode.getKey()).child("Messages").addListenerForSingleValueEve
nt(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
for(DataSnapshot keyNode : dataSnapshot.getChildren())
{
Message message = keyNode.getValue(Message.class); if(!message.isRead() &&
message.getSender().equals(typeUser))
dbRef.child("Dialogs").child(key).child("Messages").child(keyNode.getKey()).child("read").setVal
ue(true);
}
}
@Override
public void onCancelled(@NonNull DatabaseError databaseError) {
}
```

```
});
return;
}
}
}
@Override
public void onCancelled(@NonNull DatabaseError databaseError) {
}
});
}
}, 1000);
}
public void ChangeMail(String idUser, String mail)
{
dbRef.child("Users").child(idUser).child("mail").setValue(mail);
}
}
```

**User:**

```
package com.example.dyplom.Custom_classes;
public class User {
private          String id;
private String name;
private String surname;
private String phone;
private String mail;
private String city;
private String typeUser;
public String getId() {
return id;
}
public void setId(String id) {
this.id = id;
}
public String getCity() {
return city;
}
public void setCity(String city) {
this.city = city;
}
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public String getSurname() {
return surname;
}
public void setSurname(String surname) {
this.surname = surname;
}
```

```
public String getPhone() {
return phone;
}
public void setPhone(String phone) {
this.phone = phone;
}
public String getMail() {
return mail;
}
public void setMail(String mail) {
this.mail = mail;
}
public String getTypeUser() {
return typeUser;
}
public void setTypeUser(String typeUser) {
this.typeUser = typeUser;
}
User( String name , String surname , String phone, String mail, String city, String typeUser) {
this.name = name;
this.surname = surname;
this.phone = phone;
this.mail = mail;
this.typeUser = typeUser;
this.city = city;
}
public User() {
}
public void SetUser(User user)
{
name = user.getName();
surname = user.getSurname();
phone = user.getPhone();
mail = user.getMail();
typeUser = user.getTypeUser();
city=              user.getCity();
}
}
```

**Authorization:**

```
public class Authorization extends AppCompatActivity implements View.OnClickListener {
private FirebaseAuth mAuth;
private EditText ETlogin;
private          EditText ETpassword;
@RequiresApi(api = Build.VERSION_CODES.O)
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
mAuth = FirebaseAuth.getInstance();
ETlogin = (EditText) findViewById(R.id.emailEditText);
```

```
ETpassword = (EditText) findViewById(R.id.passEditText);
if(mAuth.getCurrentUser()!=null)
{
Toast.makeText(Authorization.this , "Welcome!",
Toast.LENGTH_SHORT).show();
Intent intent = new Intent(Authorization.this, Menu.class);
startActivity(intent);
finish();
}
findViewById(R.id.loginButton).setOnClickListener(this);
findViewById(R.id.registButton).setOnClickListener(this); this.setTheme(R.style.ChooseCity);
}
@Override
public void onClick(View v) {
if(v.getId() == R.id.loginButton)
{
if(!ETlogin.getText().toString().isEmpty() && !ETpassword.getText().toString().isEmpty())
LogIn(ETlogin.getText().toString(),ETpassword.getText().toString());
else
Toast.makeText(Authorization.this , "Enter all data!",
Toast.LENGTH_SHORT).show();
}
else if (v.getId() == R.id.registButton)
{
Intent intent = new Intent(Authorization.this, Registration.class); startActivity(intent);
}
}
public void          LogIn(String email, String password)
{
mAuth.signInWithEmailAndPassword(email, password)
.addOnCompleteListener(this, new OnCompleteListener<AuthResult>() { @Override
public void onComplete(@NonNull Task<AuthResult> task) {
if (task.isSuccessful()) {
                                    //  Sign in success, update UI with the signed-in user's information
                        Toast.makeText(Authorization.this , "Welcome!",
Toast.LENGTH_SHORT).show();
Intent intent = new Intent(Authorization.this, Menu.class);
startActivity(intent);
finish();
} else {
                                    //  If sign in fails, display a message to the user.
Toast.makeText(Authorization.this, "Authentication failed.", Toast.LENGTH_SHORT).show();
}
}
});
}

}
```

**Registration:**

```
public class Registration extends AppCompatActivity {
private EditText ETname;
```

```
private          EditText ETsurname;
private          EditText ETphone;
private          EditText ETcity;
private RadioButton RBPhoto;
private RadioButton RBClient;
SpinnerDialog city;
ArrayList<String> itemsCity = new ArrayList<>();
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_registration);
this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
ETname = (EditText) findViewById(R.id.nameEditText);
ETsurname = (EditText) findViewById(R.id.surnameEditText);
ETphone = (EditText) findViewById(R.id.phoneEditText);
ETcity = (EditText) findViewById(R.id.cityEditText);
RBPhoto = (RadioButton) findViewById(R.id.radioButton);
RBClient = (RadioButton) findViewById(R.id.radioButton2);
this.setTheme(R.style.ChooseCity);
ETname.requestFocus();
Collections.addAll(itemsCity,getResources().getStringArray(R.array.cities));
city = new SpinnerDialog(Registration.this,itemsCity,"Select city",R.style.ChooseCity,"Cancel") ;
city.setCancellable(true);
city.bindOnSpinerListener(new OnSpinerItemClick() {
@Override
public void onClick(String s, int i) {
ETcity.setText(s);
}
});
}
public void onClick(View v) {
if(!ETname.getText().toString().isEmpty() && !ETsurname.getText().toString().isEmpty()&&
!ETphone.getText().toString().isEmpty())
{
if(RBPhoto.isChecked() || RBClient.isChecked())
{
Intent intent = new Intent(Registration.this, NextRegistration.class); User user = new User();
intent.putExtra("name", (ETname.getText().toString())); intent.putExtra("surname",
ETsurname.getText().toString()); intent.putExtra("phone", ETphone.getText().toString());
intent.putExtra("city", ETcity.getText().toString());
if (RBPhoto.isChecked()) intent.putExtra("typeUser", "photographer"); else if
(RBClient.isChecked()) intent.putExtra("typeUser", "client"); startActivity(intent);
}
else
{
Toast.makeText(Registration.this, "Choose category.",
Toast.LENGTH_SHORT).show();
}
}
else
{
Toast.makeText(Registration.this, "Enter all data",
```

```
Toast.LENGTH_SHORT).show();
}
}
public void OnClickCity(View v)
{
city.showSpinerDialog();
}
}
NextRegistration:
public class NextRegistration extends AppCompatActivity { private FirebaseAuth mAuth;
private          EditText ETmail;
private          EditText ETpass;
private          EditText ETrepass;
DatabaseReference myRef;
FirebaseDatabase database;
DatabaseHelper databaseHelper;
ImageView img;
private Uri filePath;
private          final int PICK_IMAGE_REQUEST = 71;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_next_registration);
this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
ETmail = (EditText) findViewById(R.id.mailEditText);
ETpass = (EditText) findViewById(R.id.passEditText);
ETrepass = (EditText) findViewById(R.id.repassEditText);
database = FirebaseDatabase.getInstance();
mAuth = FirebaseAuth.getInstance();
databaseHelper = new DatabaseHelper();
img = (ImageView) findViewById(R.id.imageView3);
}
public void OnClickChoosePhoto(View v) {
Intent intent = new Intent();
intent.setType("image/*");
intent.setAction(Intent.ACTION_GET_CONTENT);
startActivityForResult(Intent.createChooser(intent, "Choose photo"),PICK_IMAGE_REQUEST);
}
public void onClick(View v)
{
if(!ETmail.getText().toString().isEmpty()&& !ETpass.getText().toString().isEmpty() &&
!ETrepass.getText().toString().isEmpty())
{
if( ETpass.getText().toString().equalsIgnoreCase(ETrepass.getText().toString()))
{
Reg(ETmail.getText().toString(),ETpass.getText().toString());
}
else
{
Toast.makeText(NextRegistration.this, "Repeat password.", Toast.LENGTH_SHORT).show();
}
}
```

```
else
{
Toast.makeText(NextRegistration.this, "Enter data for authorization.",
Toast.LENGTH_SHORT).show();
}
}
public void          Reg(String email, String password)
{
mAuth.createUserWithEmailAndPassword(email, password)
.addOnCompleteListener(this, new OnCompleteListener<AuthResult>() { @Override
public void onComplete(@NonNull Task<AuthResult> task) {
if (task.isSuccessful()) {
User user = new User();
Intent thisIntent = getIntent();
user.setName(thisIntent.getStringExtra("name"));
user.setSurname(thisIntent.getStringExtra("surname"));
user.setPhone(thisIntent.getStringExtra("phone"));
user.setCity(thisIntent.getStringExtra("city"));
user.setTypeUser(thisIntent.getStringExtra("typeUser"));
user.setMail(ETmail.getText().toString());
databaseHelper.CreateUser(mAuth.getUid(), user, new DatabaseHelper.DataStatusUser() {
@Override
public void DataIsLoaded(User user) {
}
@Override
public void DataIsInserted() {
if(filePath!=null) {
StorageReference mStorageRef =FirebaseStorage.getInstance().getReference();
StorageReference riversRef =mStorageRef.child("avatars/"+mAuth.getUid()+".jpg");
riversRef.putFile(filePath).addOnSuccessListener(new
OnSuccessListener<UploadTask.TaskSnapshot>() {
@Override
public void onSuccess(UploadTask.TaskSnapshot

taskSnapshot) {
Intent intent = new Intent(NextRegistration.this,Menu.class);
startActivity(intent);
Toast.makeText(NextRegistration.this, "Completed",Toast.LENGTH_SHORT).show();
}
})
.addOnFailureListener(new OnFailureListener() {
@Override
public void onFailure(@NonNull Exception exception) {
// Handle unsuccessful uploads
// ...
}
});
}
});
} else {
Toast.makeText(NextRegistration.this, "Error.",
Toast.LENGTH_SHORT).show();
```

```
}
}
});
}
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
super.onActivityResult(requestCode, resultCode, data);
if(requestCode == PICK_IMAGE_REQUEST && resultCode == RESULT_OK && data!=null
&& data.getData() != null)
{
filePath = data.getData();
try
{
Bitmap bitmap = MediaStore.Images.Media.getBitmap(getContentResolver(), filePath);
img.setImageBitmap(bitmap);
}
catch (IOException e)
{
e.printStackTrace();
}
}
}
}
```

**ProfileFragment:**

```
import java.io.File;
import java.io.IOException;
public class ProfileFragment extends Fragment {
private HomeViewModel homeViewModel;
FirebaseAuth mAuth;
TextView textView;
public View onCreateView(@NonNull LayoutInflater inflater,
ViewGroup container, Bundle savedInstanceState) {
homeViewModel =ViewModelProviders.of(this).get(HomeViewModel.class);
View root = inflater.inflate(R.layout.fragment_profile, container, false); final View rt = root;
mAuth = FirebaseAuth.getInstance();
DatabaseHelper databaseHelper = new DatabaseHelper();
databaseHelper.GetUser(mAuth.getUid(), new DatabaseHelper.DataStatusUser() { @Override
public void DataIsLoaded(User user) {
textView = (TextView) rt.findViewById(R.id.textView11);
textView.setText(user.getTypeUser().substring(0,1).toUpperCase()+user.getTypeUser().substring(1
)); textView = (TextView) rt.findViewById(R.id.textView12); textView.setText(user.getName());
textView = (TextView) rt.findViewById(R.id.textView8);
textView.setText(user.getSurname());
textView = (TextView) rt.findViewById(R.id.textView15);
textView.setText(user.getPhone());
textView = (TextView) rt.findViewById(R.id.textView16);
textView.setText(user.getCity());
textView = (TextView) rt.findViewById(R.id.textView31);
textView.setText(user.getMail());
final ImageView img = (ImageView) rt.findViewById(R.id.avatarMain); StorageReference
mStorageRef;
```

```
mStorageRef = FirebaseStorage.getInstance().getReference();
StorageReference storeRef = mStorageRef.child("avatars/"+mAuth.getUid()+".jpg");
File localFile = null;
try {
localFile = File.createTempFile("image", "jpg");
} catch (IOException e) {
e.printStackTrace();
}
final                    String path = localFile.getPath();
storeRef.getFile(localFile)
.addOnSuccessListener(new OnSuccessListener<FileDownloadTask.TaskSnapshot>() { @Override
public void onSuccess(FileDownloadTask.TaskSnapshot taskSnapshot) {
Glide.with(rt.getContext()).load(path).into(img);
}
});
}
});
return root;
}
}
```

**OrdersFragment:**

```
public class OrdersFragment extends Fragment {
private GalleryViewModel galleryViewModel;
FirebaseAuth mAuth;
ListView ordersLW;
Context mContext;
public View onCreateView(@NonNull final LayoutInflater inflater,
ViewGroup container, Bundle savedInstanceState) {
mContext =getActivity();
galleryViewModel =
ViewModelProviders.of(this).get(GalleryViewModel.class);
View root = inflater.inflate(R.layout.fragment_orders_user, container, false); mAuth =
FirebaseAuth.getInstance();
ordersLW = (ListView) root.findViewById(R.id.ordersListView); final DatabaseHelper
databaseHelper = new DatabaseHelper(); databaseHelper.ReadOrdersForUser(mAuth.getUid(), new
DatabaseHelper.DataStatusOrder() {
@Override
public void DataIsLoaded(Order order) {
}
@Override
public void ListIsLoaded(final List<Order> orders) {
Collections.reverse(orders);
CustomAdpater customAdpater = new CustomAdpater(mContext, orders);
ordersLW.setAdapter(customAdpater);
ordersLW.setOnItemClickListener(new AdapterView.OnItemClickListener() {
@Override

public void onItemClick(AdapterView<?> parent, View view, final int position, long id) { final int
pos = position;
```

```
Intent intent = new Intent(getActivity(), OderActivity.class); intent.putExtra("id",
orders.get(pos).getId()); String index = String.valueOf(position+1);
intent.putExtra("index", index);
startActivity(intent);
}
});

}
});
return root;
}
class CustomAdpater extends ArrayAdapter<Order>
{
private                Context mContext ;
private Integer pos;
private List<Order> orders = new ArrayList<>();
public CustomAdpater(@NonNull Context context, @NonNull List<Order> objects) {
super(context,0,objects);
mContext =context;
orders = objects;
}
@Override
public int getCount() {
return orders.size();
}
@NonNull
@Override
public Context getContext() {
return mContext;
}
@Nullable
@Override
public Order getItem(int position) {
return orders.get(position);
}
@Override
public int getPosition(@Nullable Order item) {
return super.getPosition(item);
}
@NonNull
@Override
public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
View listItem = convertView;
if(listItem == null)
listItem = LayoutInflater.from(mContext).inflate(R.layout.order_list_element,parent,false);
Order currentOrder = orders.get(position);
TextView orderTV = (TextView) listItem.findViewById(R.id.elOrderTV);
TextView dateTV = (TextView) listItem.findViewById(R.id.elDateTV);
TextView timeTV = (TextView) listItem.findViewById(R.id.elTimeTV);
TextView costTV = (TextView) listItem.findViewById(R.id.elCostTV);
TextView statusTV = (TextView) listItem.findViewById(R.id.elStatusTV);
```

```
TextView typeTV= (TextView) listItem.findViewById(R.id.elTypeTV); TextView adressTV=
(TextView) listItem.findViewById(R.id.elAdressTV);
final TextView unreadMsg = (TextView) listItem.findViewById(R.id.unreadMsgTV);
final LinearLayout undrLayout = (LinearLayout) listItem.findViewById(R.id.unreadLayout);
DatabaseHelper databaseHelper = new DatabaseHelper();
databaseHelper.GetCountOfUnreadMsg(currentOrder.getId(), "photographer",new
DatabaseHelper.CountMsg() {
@Override
public void DataIsLoaded(int count) {
if(count == 0){
undrLayout.setVisibility(View.INVISIBLE);
}
else
{
undrLayout.setVisibility(View.VISIBLE);
unreadMsg.setText(String.valueOf(count));
}
}
});
orderTV.setText("Order "+ (Integer)(position+1)); dateTV.setText(currentOrder.getDate());
timeTV.setText(currentOrder.getTime()); costTV.setText(currentOrder.getCost()+"₴");
typeTV.setText(currentOrder.getType()); adressTV.setText(currentOrder.getCity()+",
"+currentOrder.getAdress()); if(currentOrder.getOrderStatus().equals("process")) {
statusTV.setText("                    ");
statusTV.setTextColor(Color.parseColor("#81F16F0A"));
}
else if (currentOrder.getOrderStatus().equals("canceled"))
{
statusTV.setText("✕");
statusTV.setTextColor(Color.parseColor("#F80000"));
}
else
{
statusTV.setText("✓");

statusTV.setTextColor(Color.parseColor("#1AFF01"));
}
return listItem;
}
}
}
```

**OderActivity:**

```
public class OderActivity extends AppCompatActivity {
TextView order;
TextView status;
TextView date;
TextView time;
TextView adress;
TextView duration;
TextView cost;
```

```
TextView type;
EditText desc;
String id;
String index;
boolean confirmed;
Button btnPhotographers;
Button confirm;
boolean havePhotographer;
int rateClient;
String idPhotographer;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_oder);
this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
Bundle bundle = getIntent().getExtras(); DatabaseHelper databaseHelper = new DatabaseHelper();
date = (TextView) findViewById(R.id.dateTextView) ; time = (TextView)
findViewById(R.id.timeTextView) ; adress = (TextView) findViewById(R.id.adressTextView) ;
duration = (TextView) findViewById(R.id.durationTextView) ; cost = (TextView)
findViewById(R.id.costTextView) ; desc = (EditText) findViewById(R.id.desEditText);
type = (TextView) findViewById(R.id.typeTextView);
order = (TextView) findViewById(R.id.textView28);
status = (TextView) findViewById(R.id.textView33);
btnPhotographers = (Button) findViewById(R.id.button9);
confirm = (Button) findViewById(R.id.button5);
id = bundle.getString("id");
index = bundle.getString("index");
order.setText("Order "+index);
databaseHelper.GetOrder(id, new DatabaseHelper.DataStatusOrder() { @Override
public void DataIsLoaded(Order order) { date.setText(order.getDate());
time.setText(order.getTime()); adress.setText(order.getCity()+", "+order.getAdress());
duration.setText(order.getDuration()); cost.setText(order.getCost());
desc.setText(order.getDescription()); type.setText(order.getType());
if(order.getOrderStatus().equals("process"))
{
status.setText("                                ");
status.setTextColor(Color.parseColor("#81F16F0A"));
}
else if (order.getOrderStatus().equals("canceled"))
{
status.setText("✕");
status.setTextColor(Color.parseColor("#F80000"));
}
else
{
status.setText("✓");
confirmed = true;
status.setTextColor(Color.parseColor("#1AFF01"));
confirm.setVisibility(View.INVISIBLE);
}
if(order.getIdPhotographer()==null || order.getIdPhotographer().isEmpty()) {
btnPhotographers.setText("Photographer(not selected)");
```

```
havePhotographer = false;
confirm.setVisibility(View.INVISIBLE);
}
else
{
idPhotographer = order.getIdPhotographer();
btnPhotographers.setText("Photographer");
havePhotographer = true;
}
rateClient = order.getRateClient();
}
}
public void OnClickChangeOrder(View v)
{
Intent intent = new Intent(v.getContext(), ChangeOrder.class); intent.putExtra("id", id);
startActivity(intent);
}
public void OnClickPhotographers(View v)
{
if(!havePhotographer) {
Intent intent = new Intent(v.getContext(), ListPhotographers.class);
intent.putExtra("id", id);
startActivity(intent);
}
else
{
Intent intent = new Intent(v.getContext(), ProfileDetails.class);
intent.putExtra("id", idPhotographer);
intent.putExtra("idOrder", id);
intent.putExtra("isAccepted", true);
intent.putExtra("isConfirmed",confirmed);
intent.putExtra("rateClient", rateClient);
intent.putExtra("isClient", false);
startActivity(intent);
}
}
public void OnClickDeleteOrder(View v)
{
final View view = v;
AlertDialog.Builder altdial = new AlertDialog.Builder(v.getContext()); altdial.setMessage("Do you
want to delete this order?").setCancelable(false)
.setPositiveButton("Yes", new DialogInterface.OnClickListener() { @Override
public void onClick(DialogInterface dialog, int which) { DatabaseHelper databaseHelper = new
DatabaseHelper(); databaseHelper.DeleteOrder(id); Toast.makeText(view.getContext(), "Deleted",
Toast.LENGTH_SHORT).show();
onBackPressed();
}
})
.setNegativeButton("No", new DialogInterface.OnClickListener() { @Override
public void onClick(DialogInterface dialog, int which) {
dialog.cancel();
}
```

```
});
AlertDialog alertDialog = altdial.create();
alertDialog.setTitle("Warning!");
alertDialog.show();
}
public void OnClickConfirmOrder(View v)
{
final View view = v;
if(!havePhotographer)
{
Toast.makeText(view.getContext(), "Can`t confirm without photographer",
Toast.LENGTH_SHORT).show();
return;
}
AlertDialog.Builder altdial = new AlertDialog.Builder(v.getContext()); altdial.setMessage("Is order
completed?").setCancelable(false)
.setPositiveButton("Yes", new DialogInterface.OnClickListener() { @Override
public void onClick(DialogInterface dialog, int which) { DatabaseHelper databaseHelper = new
DatabaseHelper(); databaseHelper.ChangeStatusOrder(id, "completed");
Toast.makeText(view.getContext(), "Confirmed",
Toast.LENGTH_SHORT).show();
onBackPressed();
}
})
.setNegativeButton("No", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
dialog.cancel();
}
});
AlertDialog alertDialog = altdial.create();
alertDialog.setTitle("Warning!");
alertDialog.show();
}
@Override
public void onBackPressed() {
finish();
}
}
```

**AddOrder:**

```
public class AddOrder extends AppCompatActivity {
EditText date;
EditText time;
EditText duration;
EditText adress;
EditText cost;
EditText description;
FirebaseAuth mAuth;
Spinner type;
SpinnerDialog city;
ArrayList<String> itemsCity = new ArrayList<>();
```

```
TextView cityTW;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_add_order);
this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
date = (EditText) findViewById(R.id.dateEditText);
time = (EditText) findViewById(R.id.timeEditText);
duration = (EditText) findViewById(R.id.durEditText);
adress = (EditText) findViewById(R.id.adressEditText);
cost = (EditText) findViewById(R.id.costEditText);
description = (EditText) findViewById(R.id.desEditText);
mAuth = FirebaseAuth.getInstance();
AlertDialog.Builder builder;
type = (Spinner) findViewById(R.id.spinner);
ArrayAdapter<String> myAdapter = new ArrayAdapter<String>(AddOrder.this,
R.layout.element_type, getResources().getStringArray(R.array.typephotos));
myAdapter.setDropDownViewResource(R.layout.support_simple_spinner_dropdown_item);
type.setAdapter(myAdapter);
this.setTheme(R.style.ChooseCity);
cityTW = (TextView) findViewById(R.id.cityTW);
Collections.addAll(itemsCity,getResources().getStringArray(R.array.cities));
city = new SpinnerDialog(AddOrder.this,itemsCity,"Select city",R.style.ChooseCity,"Cancel") ;
city.setCancellable(true);
city.bindOnSpinerListener(new OnSpinerItemClick() {
@Override
public void onClick(String s, int i) {
cityTW.setText(s);
}
});
DatabaseHelper databaseHelper = new DatabaseHelper(); databaseHelper.GetUser(mAuth.getUid(),
new DatabaseHelper.DataStatusUser() {
@Override
public void DataIsLoaded(User user) {
cityTW.setText(user.getCity());
}
});
}
public void OnClickDate(View v)
{
final Calendar c = Calendar.getInstance();
int mYear = c.get(Calendar.YEAR);
int mMonth = c.get(Calendar.MONTH);
int mDay = c.get(Calendar.DAY_OF_MONTH);
DatePickerDialog datePickerDialog = new DatePickerDialog(this,R.style.DatePickerDialog, new
DatePickerDialog.OnDateSetListener() {
@Override
public void onDateSet(DatePicker view, int year,
int monthOfYear, int dayOfMonth) {
date.setText(dayOfMonth+"."+(int)(monthOfYear+1)+"."+year);
}
}, mYear, mMonth, mDay);
```

```
datePickerDialog.show();
}
public void OnClickTime(View v)
{
final Calendar c = Calendar.getInstance();
int mHour = c.get(Calendar.HOUR_OF_DAY);
int mMinute = c.get(Calendar.MINUTE);
// Launch Time Picker Dialog
TimePickerDialog timePickerDialog = new TimePickerDialog(this,R.style.DatePickerDialog, new
TimePickerDialog.OnTimeSetListener() {
@Override
public void onTimeSet(TimePicker view, int hourOfDay,int minute) {
time.setText(hourOfDay+":"+minute);
}
}, mHour, mMinute, false);
timePickerDialog.show();
}
public void OnClickDuration(View v)
{
final Dialog d = new Dialog(AddOrder.this);
d.setTitle("NumberPicker");
d.setContentView(R.layout.pick_int_value);
Button b1 = (Button) d.findViewById(R.id.button1);
final NumberPicker np = (NumberPicker) d.findViewById(R.id.numberPicker1);
np.setMaxValue(24);
np.setMinValue(1);
np.setWrapSelectorWheel(false);
b1.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
TextView hours = (TextView) findViewById(R.id.textViewHours);
int value = np.getValue();
if(value==1) hours.setText("hour");
else hours.setText("hours");
duration.setText(String.valueOf(np.getValue()));
d.dismiss();
}
});
d.show();
}
public void OnClickCity(View v)
{
city.showSpinerDialog();
}
public void OnClickAdd(View v)
{
boolean part1 = date.getText().toString().isEmpty() || time.getText().toString().isEmpty();
part1 = part1 || duration.getText().toString().isEmpty() || adress.getText().toString().isEmpty(); part1
= part1 || cost.getText().toString().isEmpty(); if(!part1) {
Order order = new Order();
order.setDate(date.getText().toString());
order.setTime(time.getText().toString());
```

```
order.setDuration(duration.getText().toString());
order.setAdress(adress.getText().toString());
order.setCost(cost.getText().toString());
if(description.getText().toString().isEmpty()) order.setDescription("");
else order.setDescription(description.getText().toString());
order.setOrderStatus("process");
order.setIdClient(mAuth.getUid());
order.setType(type.getSelectedItem().toString());
order.setCity(cityTW.getText().toString()); DatabaseHelper databaseHelper =new
DatabaseHelper(); databaseHelper.AddOrder(order, new DatabaseHelper.DataStatusOrder() {
@Override
public void DataIsInserted() {
Toast.makeText(AddOrder.this, "Order has created",
Toast.LENGTH_SHORT).show();
onBackPressed();
}
});
}
else
{
Toast.makeText(AddOrder.this, "Fill in all the fields",Toast.LENGTH_SHORT).show();
}
}
}
```

**DialogRoom:**

```
public class DialogRoom extends AppCompatActivity {
ListView messagesLW;
EditText writeMessage;
String idPhotographer;
String idOrder;
String idRoomDialog;
String typeUser;
String nameSurname;
TextView name;
LinearLayout head;
boolean canRead;
DatabaseHelper databaseHelper = new DatabaseHelper();
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_dialog);
this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
Bundle bundle = getIntent().getExtras();
messagesLW = (ListView) findViewById(R.id.dialogLW);
writeMessage = (EditText) findViewById(R.id.editText2);
name = (TextView) findViewById(R.id.textView56);
idPhotographer = bundle.getString("idPhotographer");
idOrder = bundle.getString("idOrder");
typeUser = bundle.getString("typeUser");
nameSurname = bundle.getString("name");
head = (LinearLayout) findViewById(R.id.linearLayout14);
```

```
canRead = true;
name.setText(nameSurname);
databaseHelper.GetDialogRoom(idPhotographer, idOrder, new DatabaseHelper.DataStatusDialog()
{ @Override
public void ListIsLoaded(List<Message> messages, String idRoom) { idRoomDialog = idRoom;
Log.d("Info", ""+messages.size());
final List<Message> copy = messages;
CustomAdpater customAdpater = new CustomAdpater(DialogRoom.this, copy);
messagesLW.setAdapter(customAdpater); if(canRead) {
if (typeUser.equals("client")) {
databaseHelper.ReadMessages(idOrder, idPhotographer, "photographer");
} else
databaseHelper.ReadMessages(idOrder, idPhotographer, "client");
}
}
});
}
public void OnClickSendMessage(View v)
{
if(!writeMessage.getText().toString().isEmpty())
{
Calendar calendar = Calendar.getInstance(); SimpleDateFormat time = new
SimpleDateFormat("HH:mm"); SimpleDateFormat date = new SimpleDateFormat("dd.MM.yyyy");
String timeStr = time.format(Calendar.getInstance().getTime());

String dateStr= date.format(Calendar.getInstance().getTime()).toString(); timeStr =
String.valueOf(Integer.valueOf(timeStr.substring(0,timeStr.indexOf(':')))+3)+timeStr.substring(tim
eStr.indexOf (':'));
databaseHelper.SendMessage(idRoomDialog, new Message(writeMessage.getText().toString(),
timeStr, dateStr, typeUser, false));
writeMessage.setText("");
}
}
class CustomAdpater extends ArrayAdapter<Message>
{
private Context mContext ;
private Integer pos;
private List<Message> messages = new ArrayList<>();
public CustomAdpater(@NonNull Context context, @NonNull List<Message> objects) {
super(context,0,objects);
mContext =context;
messages = objects;
}
@Override
public int getCount() {
return messages.size();
}
@NonNull
@Override
public Context getContext() {
return mContext;
}
```

```
@Nullable
@Override
public Message getItem(int position) {
return messages.get(position);
}
@Override
public int getPosition(@Nullable Message item) {
return super.getPosition(item);
}
@NonNull
@Override
public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
View listItem = convertView;
if(listItem == null)
listItem = LayoutInflater.from(mContext).inflate(R.layout.chat_message,parent,false); Message
current = messages.get(position);
TextView timeLeft = (TextView) listItem.findViewById(R.id.messageTimeLeft);
TextView timeRight= (TextView)          listItem.findViewById(R.id.messageTimeRight);
TextView dateLeft = (TextView)          listItem.findViewById(R.id.messageDateLeft);
TextView dateRight = (TextView)          listItem.findViewById(R.id.messageDateRight);
EditText messageLeft = (EditText) listItem.findViewById(R.id.messageTextLeft);
EditText messageRight = (EditText) listItem.findViewById(R.id.messageTextRight);
LinearLayout left = (LinearLayout)          listItem.findViewById(R.id.ChatLeft);
LinearLayout right = (LinearLayout)          listItem.findViewById(R.id.ChatRight);
LinearLayout layoutMsg = (LinearLayout)          listItem.findViewById(R.id.layoutMsg);
if(!current.getSender().equals(typeUser))
{
left.setVisibility(View.VISIBLE);
right.setVisibility(View.INVISIBLE);
timeLeft.setText(current.getTime());
dateLeft.setText(current.getDate());
messageLeft.setText(current.getText());
timeRight.setText("");
dateRight.setText("");
messageRight.setText("");
if(!current.isRead()) layoutMsg.setBackgroundColor(Color.rgb(215, 230, 232)); else
layoutMsg.setBackgroundColor(Color.rgb(255, 255, 255));
}
else
{
right.setVisibility(View.VISIBLE);
left.setVisibility(View.INVISIBLE);
timeRight.setText(current.getTime());
dateRight.setText(current.getDate());
messageRight.setText(current.getText());
timeLeft.setText("");
dateLeft.setText("");
messageLeft.setText("");
}
return listItem;
}
}
```

```
@Override
public void onBackPressed() {
canRead = false;
super.onBackPressed();
}
}
```

**FreeFragment:**

```
public class FreeFragment extends Fragment {
private SlideshowViewModel slideshowViewModel;
FirebaseAuth mAuth;
ListView ordersLW;
Context mContext;
Button filterBtn;
Filter filter;
final List<Order> listOrders = new ArrayList<>(); final List<Order> clearListOrders = new
ArrayList<>(); public View onCreateView(@NonNull LayoutInflater inflater,
ViewGroup container, Bundle savedInstanceState) {
slideshowViewModel =
ViewModelProviders.of(this).get(SlideshowViewModel.class);
View root = inflater.inflate(R.layout.fragment_slideshow, container, false); mContext
=getActivity();
mAuth = FirebaseAuth.getInstance();
ordersLW = (ListView) root.findViewById(R.id.freeOrdersList); filterBtn = (Button)
root.findViewById(R.id.button17);
filterBtn.setOnClickListener(new View.OnClickListener() { @Override
public void onClick(View v) {
Intent intent = new Intent(getActivity(), FilterOrders.class);
if(filter!=null)
{
intent.putExtra("oldFilter", true);
try {
if(filter.getStartDate()!= null)
intent.putExtra("startDate",new
SimpleDateFormat("dd.MM.yyyy").format(filter.getStartDate())); else
intent.putExtra("startDate", "");
if(filter.getEndDate()!=null)
intent.putExtra("endDate",new SimpleDateFormat("dd.MM.yyyy").format(
filter.getEndDate()));
else
intent.putExtra("endDate", "");
if(filter.getStartTime()!=null)
intent.putExtra("startTime", new
SimpleDateFormat("HH:mm").format(filter.getStartTime())); else
intent.putExtra("startTime", "");
if(filter.getEndTime()!=null)
intent.putExtra("endTime",                                    new
SimpleDateFormat("HH:mm").format(filter.getEndTime()));
else
intent.putExtra("endTime", "");
}catch (Exception e)
{
```

```
}
intent.putExtra("city", filter.getCity()); intent.putExtra("startDuration", filter.getStartDuration());
intent.putExtra("endDuration", filter.getEndDuration()); intent.putExtra("startCost",
filter.getStartCost()); intent.putExtra("endCost", filter.getEndCost());
}
else
intent.putExtra("oldFilter", false);
startActivityForResult(intent,1);
}
});
final DatabaseHelper databaseHelper = new DatabaseHelper();
databaseHelper.ReadFreeOrders(mAuth.getUid(), new DatabaseHelper.DataStatusOrder() {
@Override
public void DataIsLoaded(Order order) {
}
@Override
public void ListIsLoaded(final List<Order> orders) {
databaseHelper.ReadNotAcceptedOrders(orders, mAuth.getUid(), new
DatabaseHelper.DataStatusOrder() {
@Override
public void DataIsLoaded(Order order) {
}
@Override
public void ListIsLoaded(final List<Order> orders) {
listOrders.clear();
if(filter != null)
{
for(int i = 0; i < orders.size(); i++)
{
if(filter.isOrderFit(orders.get(i)))
listOrders.add(orders.get(i));
clearListOrders.add(orders.get(i));
}
}
else
{
for(int i = 0; i < orders.size(); i++)
{
listOrders.add(orders.get(i));
clearListOrders.add(orders.get(i));
}
}
FreeFragment.CustomAdpater customAdpater = new FreeFragment.CustomAdpater(mContext,
listOrders);
ordersLW.setAdapter(customAdpater);
ordersLW.setOnItemClickListener(new AdapterView.OnItemClickListener() {
@Override
public void onItemClick(AdapterView<?> parent, View view, final int position, long
id) {
final int pos = position;
Intent intent = new Intent(getActivity(), OrderForPhotographer.class); intent.putExtra("id",
orders.get(pos).getId()); intent.putExtra("isSubscribed", false); startActivity(intent);
```

```
}
});
}
});
}
});
return root;
}
void UpdateList()
{
Log.d("INFO", "Updating");
Log.d("INFO", String.valueOf(listOrders.size()));
Log.d("INFO", String.valueOf(clearListOrders.size()));
if(filter != null)
{
listOrders.clear();
for(int i = 0; i < clearListOrders.size(); i++)
{
listOrders.add(clearListOrders.get(i));
}
for(int i = 0; i < listOrders.size(); i++)
{
if(!filter.isOrderFit(listOrders.get(i))) {
Log.d("INFO", "removing");
listOrders.remove(i);
i--;
}
}
FreeFragment.CustomAdpater customAdpater = new FreeFragment.CustomAdpater(mContext,
listOrders);
ordersLW.setAdapter(customAdpater);
}
}
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) { if(data==null) return;
try {
Date startDate;
if(!data.getStringExtra("startDate").isEmpty())
startDate =new SimpleDateFormat("dd.MM.yyyy").parse(data.getStringExtra("startDate")); else
startDate = null;
Date endDate;
if(!data.getStringExtra("endDate").isEmpty())
endDate = new SimpleDateFormat("dd.MM.yyyy").parse(data.getStringExtra("endDate")); else
endDate = null;
Date startTime;
if(!data.getStringExtra("startTime").isEmpty())
startTime = new SimpleDateFormat("HH:mm").parse(data.getStringExtra("startTime")); else
startTime = null;
Date endTime;
if(!data.getStringExtra("endTime").isEmpty())
endTime = new SimpleDateFormat("HH:mm").parse(data.getStringExtra("endTime")); else
endTime = null;
```

```
String city = data.getStringExtra("city");
int startDuration = data.getIntExtra("startDuration",0);
int endDuration = data.getIntExtra("endDuration",0);;
double startCost = data.getDoubleExtra("startCost", 0);
double endCost = data.getDoubleExtra("endCost", 0);
filter = new Filter(startDate, endDate, startTime, endTime, city, startDuration, endDuration,
startCost, endCost);
UpdateList();
}
catch (Exception e)
{
e.printStackTrace();
Log.e("Error","dd ");
}
super.onActivityResult(requestCode, resultCode, data);
}
class CustomAdpater extends ArrayAdapter<Order>
{
private Context mContext ;
private Integer pos;
private List<Order> orders = new ArrayList<>();
public CustomAdpater(@NonNull Context context, @NonNull List<Order> objects) {
super(context,0,objects);
mContext =context;
orders = objects;
}
@Override
public int getCount() {
return orders.size();
}
@NonNull
@Override
public Context getContext() {
return mContext;
}
@Nullable
@Override
public Order getItem(int position) {
return orders.get(position);
}
@Override
public int getPosition(@Nullable Order item) {
return super.getPosition(item);
}
@NonNull
@Override
public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
View listItem = convertView;
if(listItem == null)
listItem =
LayoutInflater.from(mContext).inflate(R.layout.free_orders_list_element,parent,false);
Order currentOrder = orders.get(position);
```

```
TextView dateTV = (TextView) listItem.findViewById(R.id.dateFreeOrder); TextView timeTV =
(TextView) listItem.findViewById(R.id.timeFreeOrder); TextView costTV = (TextView)
listItem.findViewById(R.id.costFreeOrder); TextView typeTV= (TextView)
listItem.findViewById(R.id.typeFreeOrder); TextView adressTV= (TextView)
listItem.findViewById(R.id.adressFreeOrder); dateTV.setText(currentOrder.getDate());
timeTV.setText(currentOrder.getTime()); costTV.setText(currentOrder.getCost()+"₴");
typeTV.setText(currentOrder.getType()); adressTV.setText(currentOrder.getCity()+",
"+currentOrder.getAdress()); final ImageView icon = (ImageView)
listItem.findViewById(R.id.avatarClient);
StorageReference mStorageRef;
mStorageRef = FirebaseStorage.getInstance().getReference();
StorageReference storeRef = mStorageRef.child("avatars/"+currentOrder.getIdClient()+".jpg");
File localFile = null;
try {
localFile = File.createTempFile("image", "jpg");
} catch (IOException e) {
e.printStackTrace();
}
final                String path = localFile.getPath();
storeRef.getFile(localFile)
.addOnSuccessListener(new OnSuccessListener<FileDownloadTask.TaskSnapshot>() { @Override
public void onSuccess(FileDownloadTask.TaskSnapshot taskSnapshot) {
Glide.with(getContext()).load(path).into(icon);
}
});
return listItem;
}
}
}
```

**SettingsFragment:**

```
public class SettingsFragment extends Fragment {
private ShareViewModel shareViewModel;
Button btnChangeMail, btnChangePassword;
EditText mailTV, userPassword ,passwordTV, confPasswordTV; public View
onCreateView(@NonNull LayoutInflater inflater,
ViewGroup container, Bundle savedInstanceState) {
shareViewModel =
ViewModelProviders.of(this).get(ShareViewModel.class);
View root = inflater.inflate(R.layout.fragment_share, container, false);
shareViewModel.getText().observe(this, new Observer<String>() {
@Override
public void onChanged(@Nullable String s) {
}
});
final FirebaseAuth mAuth = FirebaseAuth.getInstance(); final DatabaseHelper databaseHelper =
new DatabaseHelper(); btnChangeMail = (Button) root.findViewById(R.id.button22);
btnChangePassword = (Button) root.findViewById(R.id.button23); mailTV = (EditText)
root.findViewById(R.id.editText); passwordTV = (EditText) root.findViewById(R.id.editText4);
confPasswordTV = (EditText) root.findViewById(R.id.editText5); userPassword = (EditText)
root.findViewById(R.id.editText6);
final FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
```

```
btnChangeMail.setOnClickListener(new View.OnClickListener() { @Override
public void onClick(View v) {
if(!mailTV.getText().toString().isEmpty())
{
if(userPassword.getText().toString().isEmpty())
{
Toast.makeText(getActivity(), "Enter password",
Toast.LENGTH_SHORT).show();
return;
}
AuthCredential credential = EmailAuthProvider
.getCredential(user.getEmail(), userPassword.getText().toString());
user.reauthenticate(credential).addOnCompleteListener(new OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if(task.isSuccessful()) {

mAuth.getCurrentUser().updateEmail(mailTV.getText().toString()).addOnCompleteListener(new
OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if (task.isSuccessful()) {
databaseHelper.ChangeMail(mAuth.getUid(),mailTV.getText().toString());
Toast.makeText(getActivity(), "Email was changed",Toast.LENGTH_SHORT).show();
mailTV.setText("");
userPassword.setText("");
} else
Toast.makeText(getActivity(), "Error",Toast.LENGTH_SHORT).show();
}
});
}
else
{
Toast.makeText(getActivity(), "Error",Toast.LENGTH_SHORT).show();
}
}
});
}
else
Toast.makeText(getActivity(), "Enter new mail",Toast.LENGTH_SHORT).show();
}
});
btnChangePassword.setOnClickListener(new View.OnClickListener() { @Override
public void onClick(View v) {
if(!passwordTV.getText().toString().isEmpty() &&
!confPasswordTV.getText().toString().isEmpty())
{
if(passwordTV.getText().toString().equals(confPasswordTV.getText().toString()))
{
if(userPassword.getText().toString().isEmpty())
{
Toast.makeText(getActivity(), "Enter password",Toast.LENGTH_SHORT).show();
```

```
return;
}
AuthCredential credential = EmailAuthProvider.getCredential(user.getEmail(),
userPassword.getText().toString());
user.reauthenticate(credential).addOnCompleteListener(new OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if(task.isSuccessful())
{
mAuth.getCurrentUser().updatePassword(passwordTV.getText().toString()).addOnCompleteListen
er(new OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if(task.isSuccessful()) {
Toast.makeText(getActivity(), "Password was changed",Toast.LENGTH_SHORT).show();
passwordTV.setText("");
confPasswordTV.setText("");
userPassword.setText("");
}
else
Toast.makeText(getActivity(), "Error",Toast.LENGTH_SHORT).show();
}
});
}
else
Toast.makeText(getActivity(), "Error",Toast.LENGTH_SHORT).show();
}
});
}
else

Toast.makeText(getActivity(), "Passwords are not equals", Toast.LENGTH_SHORT).show();
}
else
{
Toast.makeText(getActivity(), "Enter password",Toast.LENGTH_SHORT).show();
}
}
});
return root;
}
```

**ДОДАТОК Б**

**ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ**

**ДОДАТОК В**

## ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

| **Ім’я файлу** | **Опис** |
|---|---|
| Пояснювальні документи | |
| Диплом_ doc | Пояснювальна записка до кваліфікаційної роботи. Документ Word. |
| Диплом_.pdf | Пояснювальна записка до кваліфікаційної роботи в форматі PDF |
| Програма | |
| Program.rar | Архів. Містить коди програми і откомпільовану програму |
| Презентація | |
| Презентація_.ppt | Презентація кваліфікаційної роботи |