Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка мобільного застосунку для студії танців «SDC» на основі клієнт-серверної архітектури

Виконав: студент IV курсу, групи СНс-42
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

_____ Лісовський В.В.
(підпис) (прізвище та ініціали)

Керівник _____ Млинко Б.Б.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Луцків А.М.
(підпис) (прізвище та ініціали)

Тернопіль-2021

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Лісовському Владиславу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка мобільного застосунку для студії танців «SDC» на основі клієнт-серверної архітектури

Керівник роботи Млинко Богдана Богданівна, кандидат технічних наук, доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «__2__» ____березня_____ 2021 року № 4/7-824

2. Термін подання студентом завершеної роботи 24 червня 2021

3. Вихідні дані до роботи Відомості від замовника, літературні та Інтернет-джерела

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Постановка завдання на розробку мобільного застосунку для студії танців «SDC». 1.1 Аналіз предметної області. 1.1.1 Аналіз програмного забезпечення, що використовується для створення мобільних застосунків. 1.1.2 Аналіз середовищ розробки. 1.2 Постановка задачі. 1.2.1 Опис організації замовника. 1.2.2 Призначення мобільного застосунку. 1.2.3 Виділення основних сутностей мобільного застосунку. 1.2.4 Вимоги до розробки мобільного застосунку. 1.3 Висновок до першого розділу. 2 Проектування та тестування мобільного застосунку для студії танців «SDC». 2.1 Пошук актантів та варіантів використання. 2.2 Проектування архітектури мобільного застосунку для студії танців «SDC». 2.3 Опис програмних рішень. 2.3.1 Опис програмних рішень на стороні сервера. 2.3.2 Опис програмних рішень на стороні мобільного застосунку. 2.4 Опис інтерфейсу мобільного застосунку для студії танців «SDC». 2.5 Тестування та валідація мобільного застосунку для студії танців «SDC». 2.6 Висновок до другого розділу. 3 Безпека життєдіяльності, основи охорони праці. 3.1 Долікарська допомога при переломах. 3.2 Методи оцінки соціальної та соціально-економічної ефективності заходів щодо покращенню умов та охорони праці. 3.3 Вимоги до виробничих приміщень для експлуатації ВДТ. 3.4 Висновки до третього розділу. Висновки. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1 Титульна сторінка. 2 Мета і завдання. 3 Порівняльна характеристика Java та Kotlin. 4 Структура системи. 5 Варіанти використання. 6 Архітектура сервера. 7 Аріхтектура застосунку. 8 Навігаційна архітектура. 9 Граф навігації. 10 Відео-демонстрація. 11 Висновки. 12 Подяка.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О.Я., доцент кафедри МТ	28.04	10.06

7. Дата видачі завдання 25 січня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01 – 30.01	*Виконано*
2.	Підбір джерел про проектування та розробку мобільних	01.02 – 05.02	*Виконано*
	застосунків		
3.	Переклад та опрацювання джерел	08.02 – 16.02	*Виконано*
4.	Виконання дослідження щодо вибору програмних	17.02 – 26.02	*Виконано*
	засобів та архітектури		
5.	Розроблення мобільного застосунку для студії танців	01.03-31.03	*Виконано*
	«SDC» на основі клієнт-серверної архітектури		
6.	Оформлення розділу першого розділу « Постановка	03.05 – 07.05	*Виконано*
	завдання на розробку мобільного застосунку для студії		
	танців «SDC» на основі клієнт-серверної архітектури»		
7.	Оформлення другого розділу « Проектування та	10.05 – 14.05	*Виконано*
	тестування мобільного застосунку для студії танців		
	«SDC» на основі клієнт-серверної архітектури»		
8.	Виконання завдання до підрозділу «Безпека життєдіяльності»	17.05 – 19.05	*Виконано*
9.	Виконання завдання до підрозділу «Основи охорони	19.05 – 21.05	*Виконано*
	праці»		
10.	Оформлення кваліфікаційної роботи	24.05 – 02.06	*Виконано*
11.	Нормоконтроль	04.06 – 09.06	*Виконано*
12.	Перевірка на плагіат	10.06	*Виконано*
13.	Попередній захист кваліфікаційної роботи	23.06	*Виконано*
14.	Захист кваліфікаційної роботи	24.06	

Студент _____ Лісовський В.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Млинко Б. Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка мобільного застосунку для студії танців «SDC» на основі клієнт-серверної архітектури // Кваліфікаційна робота освітнього рівня «Бакалавр» // Лісовський Владислав Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНс-42 // Тернопіль, 2021 // С. 63, рис. – 25 , табл. – 1, кресл. – 0, додат. – 63, бібліогр. – 30.

Ключові слова: kotlin, мобільний застосунок, android, клієнт-серверна архітектура, navigation architecture, студія танців «SDC», python.

Кваліфікаційна робота присвячена розробці мобільного застосунку для студії танців «SDC» на основі клієнт-серверної архітектури під платформу Android, використовуючи мову програмування Kotlin.

Метою кваліфікаційної роботи є розробка мобільного застосунку для студії танців «SDC» на основі клієнт-серверної архітектури.

В першому розділі дипломної роботи розглянута предметна область роботи, програмне забезпечення та середовища розробки, та поставлене завдання з описом організації замовника, виділенням основних сутностей та складенням вимог до застосунку.

В другому розділі дипломної роботи розглянуті варіанти використання та наявні актори, проведено процес проектування архітектури та розробки застосунку, описані створенні програмні рішення як мобільного застосунку, так і сервера, разом з описом інтерфейсу, і здійснене тестування та валідація мобільного застосунку для студії танців «SDC».

Основні результати: розроблений мобільний застосунок для студії танців «SDC» та пояснювальна записка.

ANNOTATION

Client-server architecture-based mobile application development for dance studio «SDC» // Qualification work of Bachelor educational degree // Lisovskyi Vladyslav // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, SNs-42 group // Ternopil, 2021 // Pages – 63, figures – 25, tables – 1, sketches – 0, addendums – 63, references – 30.

Keywords: kotlin, mobile application, android, client-server architecture, navigation architecture, dance club «SDC», python.

The qualification work is devoted to developing a mobile application for the dance studio "SDC" based on the client-server architecture for the Android platform, using the programming language Kotlin.

In the first section of the qualification work, the subject area of the work, software, and development environment are considered. The task with the description of the customer organization, selection of the central entities, and drafting of the requirements to the application is set.

The second section of the qualification work reviews the use cases and available actors, the process of architecture design and application development, describes the creation of software solutions for both mobile application and server, along with a description of the interface and provides testing and validation of a mobile application for dance studio "SDC".

The main result is a developed mobile application for the dance studio "SDC" and an explanatory note.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Мобільний застосунок (також відомий як додаток) – програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях.

SDC (англ. Social Dance Club) – Клуб соціальних танців.

БД – База даних.

QR (англ. Quick Response) – матричний код.

HTML (англ. Hyper text markup language) – мова розмітки гіпертекстових документів.

CSS (англ. Cascading style sheets) – каскадні таблиці стилів.

JS (англ. JavaScript) – мова програмування JavaScript.

ООП – об’єктно-орієнтоване програмування.

API (англ. Application Programming Interface) – прикладний програмний інтерфейс.

JSON (англ. JavaScript Object Notation) – запис об’єктів JavaScript, текстовий формат обміну даними.

ВДТ – відео-дисплейний термінал.

SDK (англ. Software Development Kit) – набір із засобів розробки, утиліт і документації.

SSL (англ. Secure Sockets Layer) – рівень захищених сокетів.

МП – мова програмування.

ЗМІСТ

ВСТУП 8
РОЗДІЛ 1. ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ СТУДІЇ ТАНЦІВ «SDC» 9
1.1 Аналіз предметної області 9
1.1.1 Аналіз програмного забезпечення, що використовується для створення мобільних застосунків 9
1.1.2 Аналіз середовищ розробки 12
1.2 Постановка задачі 13
1.2.1 Опис організації замовника 13
1.2.2 Призначення мобільного застосунку 14
1.2.3 Виділення основних сутностей мобільного застосунку 15
1.2.4 Вимоги до розробки мобільного застосунку 15
1.3 Висновок до першого розділу 17
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ СТУДІЇ ТАНЦІВ «SDC» 18
2.1 Пошук актантів та варіантів використання 18
2.2 Проектування архітектури мобільного застосунку для студії танців «SDC» 20
2.3 Опис програмних рішень 23
2.3.1 Опис програмних рішень на стороні сервера 23
2.3.2 Опис програмних рішень на стороні мобільного застосунку 28
2.4 Опис інтерфейсу мобільного застосунку для студії танців «SDC» 33
2.5 Тестування та валідація мобільного застосунку для студії танців «SDC» 46
2.6 Висновок до другого розділу 51
РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 52

3.1 Долікарська допомога при переломах 52
3.2 Методи оцінки соціальної та соціально-економічної ефективності заходів щодо покращенню умов та охорони праці 56
3.3 Вимоги до виробничих приміщень для експлуатації ВДТ 57
3.4 Висновки до третього розділу 59
ВИСНОВКИ 60
ПЕРЕЛІК ДЖЕРЕЛ 61
ДОДАТКИ

ВСТУП

Актуальність теми. Внаслідок широкого поширення мобільних пристроїв стало можливим проведення автоматизації рутинних завдань обліку та звітностей. Оскільки мобільні пристрої присутні практично в кожної людини – питання автоматизації обліку з використанням смартфонів, що дозволить спростити та полегшити роботу користувачів, стало особливо актуальним. Саме заради автоматизації процесу обліку та оцифрування наявних даних було запропонована розробка мобільного застосунку для студії танців «SDC».

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є розробка мобільного застосунку для студії танців «SDC» на основі клієнт-серверної архітектури. Для досягнення поставленої мети необхідно розв’язати наступні задачі:

– Дослідити доступні програмні інструменти для розробки мобільного застосунку.

– Сформувати постановку задачі – розглянути організацію замовника, поставлені вимоги до розробки мобільного застосунку та виділити основні сутності, що присутні в мобільному застосунку.

– Провести аналіз варіантів використання та архітектури застосунку.

– Розробити мобільний застосунок для студії танців «SDC» на основі клієнт-серверної архітектури.

– Провести опис реалізованих програмних рішень як клієнтської, так і серверної частини.

– Провести тестування та валідацію мобільного застосунку для студії танців «SDC».

РОЗДІЛ 1. ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ СТУДІЇ ТАНЦІВ «SDC»

1.1 Аналіз предметної області

1.1.1 Аналіз програмного забезпечення, що використовується для створення мобільних застосунків

Першим кроком при постановці завдання є аналіз та вибір засобів реалізації мобільного застосунку, тому що існує безліч технологій, які дозволять вирішити поставлену проблему, зі своїми перевагами та недоліками.

В першу чергу варто виділити три основні типи мобільних застосунків, від вибору якого буде залежати мова програмування проекту – веб, нативні та гібридні застосунки [26].

Веб-застосунок використовує у розробці веб-технології – HTML, CSS та JS. Це дозволить застосунку використовуватись на багатьох платформах, що спростить етап розробки, проте в них немає можливості використання операційної системи на повну разом з апаратними ресурсами [6].

Нативні застосунки розробляються спеціально під платформу мобільного пристрою. Саме завдяки цьому в них найбільша швидкість роботи та найширший можливий функціонал. Їх мінус полягає в тому, що для роботи додатку на іншій платформі – необхідно переписувати застосунок повністю з використанням іншої мови програмування. Для розробки нативних додатків під платформу Android використовують Java або Kotlin, під платформу IOS – Swift або Objective C.

Гібридні застосунки поєднують в собі обидва попередніх типа – веб та нативний. Вони використовують в розробці видозмінені веб-технології, але

мають набагато ширший доступний функціонал, який, проте, все ще поступає нативним [27, 28, 29].

Оскільки, мобільний застосунок для студії танців «SDC» повинен бути ефективним, тобто швидким і використовувати всі допустимі можливості операційної системи та апаратного забезпечення, і поки потреби у кросплатформеності не має, необхідно розробити додаток лише під платформу Android, найдоцільніше використовувати нативний тип розробки.

Так як, для нативної розробки під платформу Android використовуються дві мови програмування, варто розглянути їх переваги та недоліки для вибору кращої.

Java – одна з найпопулярніших об’єктно-орієнтованих мов сьогодення. Це зумовлено тим, що Java однією з перших почала використовувати ООП, і має власну віртуальну машину – спеціальне середовище, де виконуються та працюють скрипти [10]. В лістингу 1.1 продемонстрований приклад коду на Java.

Лістинг 1.1 – Приклад коду на Java

```
import com.vlad.lisovskyi.demo;
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
           System.out.println(args);
    }
}
```

Kotlin, в порівнянні з Java – доволі молода мова програмування. Вона схожа на Java, оскільки по своїй суті є надбудовою, що працює поверхи віртуальної машини Java, тим самим має всі переваги Java. Основні відмінності полягають у змінах в синтаксисі. Kotlin спроектований таким чином, щоб бути простішим в сприйнятті, ніж Java, а також безпечним і зручним [12, 13]. В лістингу 1.2 продемонстрований приклад коду на Kotlin.

Лістинг 1.2 – Приклад коду на Kotlin

```
package com.vlad.lisovskyi.demo
fun main(args: Array<String>) {
    println(args.contentToString())
    var hello: String? = “Hello, world!”
    println(hello)
}
```

Kotlin не є повністю об’єктно-орієнтованим, це статично-типізована мультипарадигмова мова програмування, яка також підтримує функційне програмування.

Весь код, написаний на одній з вищеназваних мов програмування, можна без зайвих проблем перетворити в код іншої мови.

В таблиці 1.1 зведена порівняльна характеристика цих мов [11, 14].

Таблиця 1.1 – Порівняльна характеристика Java та Kotlin

Java	Kotlin
Відсутність нуль-безпеки, всі випадки появи пустих значень мають ловитись в try-catch блоках	Всі значення за замовчуванням не є пустими, для оголошення пустих значень використовують символ «?»
Лямбда вирази з’являються лише в пізніших версіях	Лямбда вирази доступні за замовчуванням
Функційне програмування з’являється лише в пізніших версіях	Функційне програмування доступне за замовчуванням
Перевірені виключення присутні	Перевірені виключення відсутні
Класи даних оголошуються як звичайні класи з переліком усіх атрибутів та методів	Класи даних можна сформувати автоматично, використавши клас з дописом data
Розширення класу проводиться шляхом створення нового класу, що успадковує бажаний з доданням нового методу	Розширення класу можна провести динамічно, використовуючи префіксне створення бажаного методу

Оскільки, Kotlin є безпечнішим, простішим та надійнішим [11, 14], а по функціоналу практично не відрізняється від Java, і з 2019 року є рекомендованою мовою для розробки під платформу Android – саме він був обраним для розробки мобільного застосунку для студії танців «SDC».

1.1.2 Аналіз середовищ розробки

Після вибору мови програмування не менш важливим кроком є вибір середовища розробки, яке забезпечить процес розробки великою кількістю допоміжних інструментів, що сильно спростять та пришвидшать роботу.

У сфері розробки мобільних застосунків найчастіше використовуються IntelliJ IDEA та Android Studio.

IntelliJ IDEA – це універсальне інтегроване середовище розробки, що використовує комерційну модель використання. Воно також має безкоштовну версію з урізаним функціоналом. IntelliJ має велику кількість додаткових інструментів та плагінів, корисних для роботи з мобільними за стосунками [8, 9]. Приклад інтерфейсу IntelliJ IDEA розміщений на рисунку 1.1.

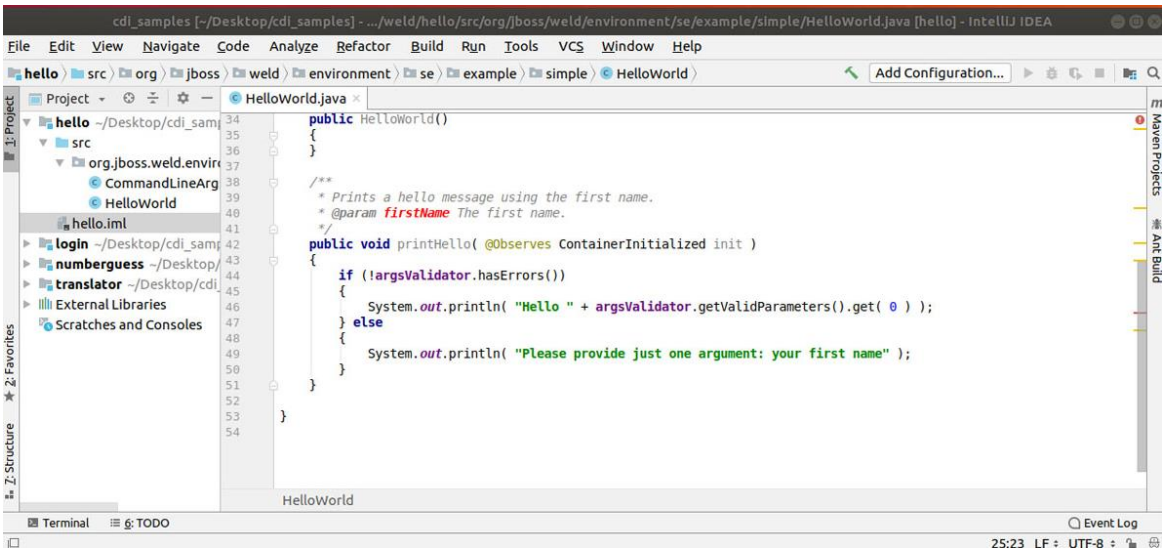


Рисунок 1.1 – Інтерфейс IntelliJ IDEA

Android Studio – вільне інтегроване середовище розробки, яке пропонується від Google для використання під час розробки мобільних застосунків з платформою Android. Воно побудоване на основі середовища IntelliJ IDEA як вільний аналог, і загалом містить в собі основні його плюси – велику кількість інтегрованих інструментів, проте лише спеціалізованих під розробку для додатків [1, 2]. Приклад інтерфейсу Android Studio зображено на рисунку 1.2.

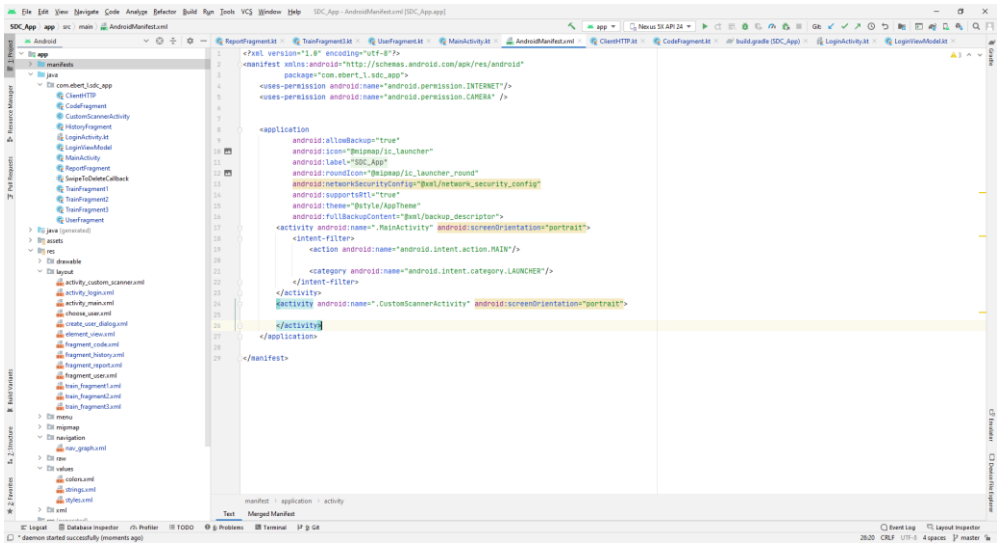


Рисунок 1.2 – Інтерфейс Android Studio

Оскільки Android Studio є більш спеціалізованим для роботи з мобільними за стосунками – саме він був обраний для розробки мобільного застосунку для студії танців «SDC».

1.2 Постановка задачі

1.2.1 Опис організації замовника

Замовником виступає студія танців Social Dance Club (див. рис. 1.3), одна з перших спеціалізованих студій танців Тернополя, що має декілька точок розміщення у місті, і користується великою популярністю.

Рисунок 1.3 – Логотип студії танців

Студія танців пропонує професійних інструкторів та великий вибір серед різновидів танців, особливо спеціалізованих латиноамериканських [16], таких як:

– сальса;
– бачата;
– кізомба;
– зук;
– реггетон та інші.

У зв’язку з великою кількістю інструкторів, учнів та програм перед студією постала необхідність у створенні мобільного застосунку, який автоматизує роботу інструкторів.

1.2.2 Призначення мобільного застосунку

Розробка мобільного застосунку студії танців «SDC» призначена для полегшення роботи інструкторів шляхом автоматизації їх взаємодії з обліком абонементів та отримання аналітичних даних.

Додаток реалізує рівень інтерфейсу між клієнтом (інструктором) та серверною частиною (рівнями обробки та даних) в трирівневій клієнт-серверній архітектурі [19, 30].

Таким чином мобільний застосунок дозволить спростити та пришвидшити такі процеси як зберігання та облік даних, завдяки автоматизованих перевірках доступів та виконання різних умов.

1.2.3 Виділення основних сутностей мобільного застосунку

На етапі проектування можна виділити наступні сутності:

– Інструктор (клієнт, який використовує мобільний застосунок; характеризується логіном та паролем).

– Учень (характеризується певним кодом).

– Абонемент (право на відвідувань тренувань, є прив’язаним до учня; характеризується терміном дії, кількістю відвідувань, ціною, номером).

– Танцювальна група (для якої будуть створюватись тренування).

– Тренування (зберігає інформацію про тренування, що відбулись, є прив’язаною до інструктора, абонемента та групи; характеризується датою та посиланнями на прив’язані сутності).

Кожна з складених сутностей описує модель даних, що зберігається на серверному рівні даних (тобто таблиця в БД).

1.2.4 Вимоги до розробки мобільного застосунку

Проаналізувавши предметну область та обговоривши функціонал з замовником, можна сформувати загальні вимоги до розробки додатку:

1. Вимоги до інтерфейсу та платформи:

– інтерфейс повинен бути зрозумілим та ненавантаженим візуальною складовою;

– логотип студії повинен відображатись у вигляді іконки мобільного застосунку;

– інтерфейс повинен використовувати сучасні компоненти;

– розробка мобільного застосунку виконується під платформу Android, мінімальна версія SDK – 23.

2. Вимоги до доступу та захисту:

– користувачі не мають прямого доступу до рівня даних, а використовують серверні функції;

– вхід користувача захищений парольною автентифікацією;

– паролі в БД зберігаються у хешованому вигляді;

– паролі відсилаються лише раз при вході в додаток, далі продовж сесії мобільний застосунок використовує тимчасовий токен автентифікації, згенерований сервером у відповідь на вхід;

– логін введений користувачем повинен зберігатись і для наступного входу;

– пароль при вводі в форму не повинен відображатись текстом.

3. Вимоги до функціональних можливостей:

– можливість створення інструктором сесії (сукупності тренувань) для певної групи і з доданням ще одного інструктора;

– можливість сканування QR-коду абонемента;

– можливість додання учнів у поточну сесію;

– можливість видалення учнів з поточної сесії;

– можливість додання одного і того самого учня в сесію декілька разів;

– можливість додання учнів без сканування QR-коду;

– можливість додання одноразового анонімного абонементу;

– створена активна сесія повинна зберігатись на сервері і бути доступною для обох інструкторів;

– можливість отримання інформації про учня через сканування QR-коду.

– можливість реєстрації нового учня як в сесії, так і поза нею, шляхом сканування QR-коду, що не існує в БД;

– можливість створення нового абонементу для учня;

– можливість перегляду всього списку учнів та зміни їх інформації;

– можливість перегляду історії тренувань для поточного абонементу учня;

– можливість зміни QR-коду на новий.

4. Вимоги до звітності:

– можливість перегляду історії проведених сесій для груп;

– можливість перегляду фінансів за певний місяць (кількості грошей у касі за куплений абонемент);

– можливість перегляду звіту з нарахування зарплати за місяць.

Розроблений мобільний застосунок повинен відповідати усім вищеперерахованим вимогам.

1.3 Висновок до першого розділу

В першому розділі кваліфікаційної роботи був проведений аналіз предметної області, тобто були досліджені доступні програмні інструменти для розробки мобільного застосунку – мови програмування та інтегровані середовища розробки. Була сформована постановка задачі – була розглянута організацію замовника, були поставлені вимоги до розробки мобільного застосунку та виділені основні сутності, що присутні в мобільному застосунку.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ СТУДІЇ ТАНЦІВ «SDC»

2.1 Пошук актантів та варіантів використання

На рисунку 2.1 зображена діаграма варіантів використання з акторами та прецедентами [17, 18], що виникнуть у розробленій клієнт-серверній системі для студії танців «SDC».

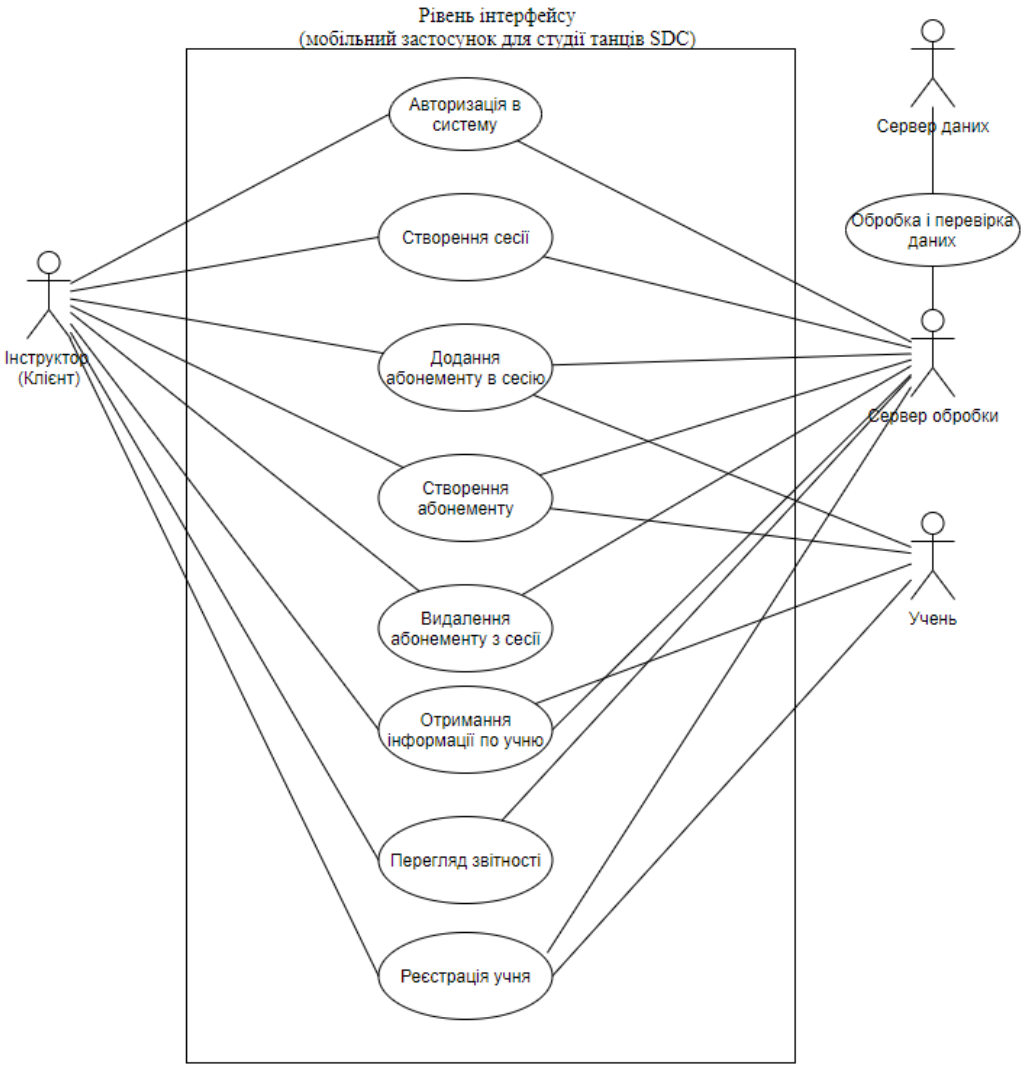


Рисунок 2.1 – Діаграма варіантів використання системи мобільного застосунку

В системі будуть виділі чотири основних актори – Інструктор або Клієнт (актор, що використовує мобільний застосунок для студії танців «SDC»), Учень (актор, який бере участь у прецедентах пов’язаних з скануванням QR-коду), Сервер обробки (актор, який виконує обробку запитів) та Сервер даних (актор, що постачає дані серверу обробки).

В якості прецедентів виступлять сформовані функціональні вимоги до додатку, тобто основні дії, які може виконати актор в рамках використання мобільного застосунку для студії танців «SDC».

Оскільки архітектура розроблюваної системи з участю мобільного застосунку для студії танців «SDC» є клієнт-серверною [19], доцільно при зображені можливих варіантів використання зобразити існування акторів, які виконують роль рівнів обробки та даних, і з якими взаємодіє Клієнт через додаток.

Сервер обробки буде проводити обробку всіх можливих дій та запитів у системі і обмінюється даними з Сервером даних.

Інструктор буде виконувати ініціалізацію можливих дій в рамках системи мобільного застосунку, інформація про які надсилаються Серверу обробки. Деякі з дій є незалежними, тобто Інструктор може ініціалізувати їх у будь-який момент власноруч:

– авторизація в систему з отриманням токену доступу;
– створення сесії для певної групи;
– видалення вже доданого абонементу з сесії;
– перегляд звітності.

Для інших дій – Створення абонементу, Додання абонементу в сесію, Реєстрація учня, Отримання інформації про учня – Інструктору може знадобитись участь Учня, тобто подання ним власного QR-коду. Проте, також будуть створені механізми, які при потребі дозволять Інструктору виконати ці прецеденти без участі Учня.

2.2 Проектування архітектури мобільного застосунку для студії танців «SDC»

Для реалізації всіх вимог до мобільного застосунку для студії танців «SDC» необхідно створити багатоекранний додаток, в якому кожен екран виконує конкретно визначені завдання. Завдяки такому розподілу інтерфейс не буде перенавантажений візуальною складовою. Були відокремлені наступні екрани:

– екран авторизації, який буде виконувати перевірку належності користувача до сутності Інструктор;

– екран домашньої сторінки, з якого буде виконуватись навігація по інших екранах;

– екран створення сесії, в якому будуть вводитись необхідні дані для створення сесії;

– екран контролю поточної сесії, в якому будуть записуватись всі абонементи та тренування;

– екран перегляду історії, який дозволить переглянути інформацію по закритих сесіях;

– екран перегляду звітності по фінансах та зарплаті, в якому можна переглянути відповідні звіти;

– екран перегляду учнів, в якому можна побачити список всіх учнів, їх поточний абонемент та змінити основні інформацію про них;

– екран сканування коду учня, який представляє детальнішу інформацію про учня та його абонемент.

Для створення багатоекранного застосунку було вирішено використати навігаційну архітектуру [15]. Компонент Навігація надає можливість зручного переходу між різними екранами, реалізовуючи функціонал зміни екранів, натиску кнопок, стеку відкритих екранів та багато інших особливостей.

Компонент Навігація складається з трьох елементів:

– контейнер NavHost, який буде відображувати кінцеві фрагменти;

– навігаційний об’єкт NavController, що буде керувати навігацією фрагментів у контейнері, та реалізовувати обмін екранів;

– граф навігації NavGraph – XML-ресурс, який містить інформацію про існуючі екрани, та можливі шляхи проходження скрізь них. Граф навігації мобільного застосунку для студії танців «SDC» зображений на рисунку 2.2.

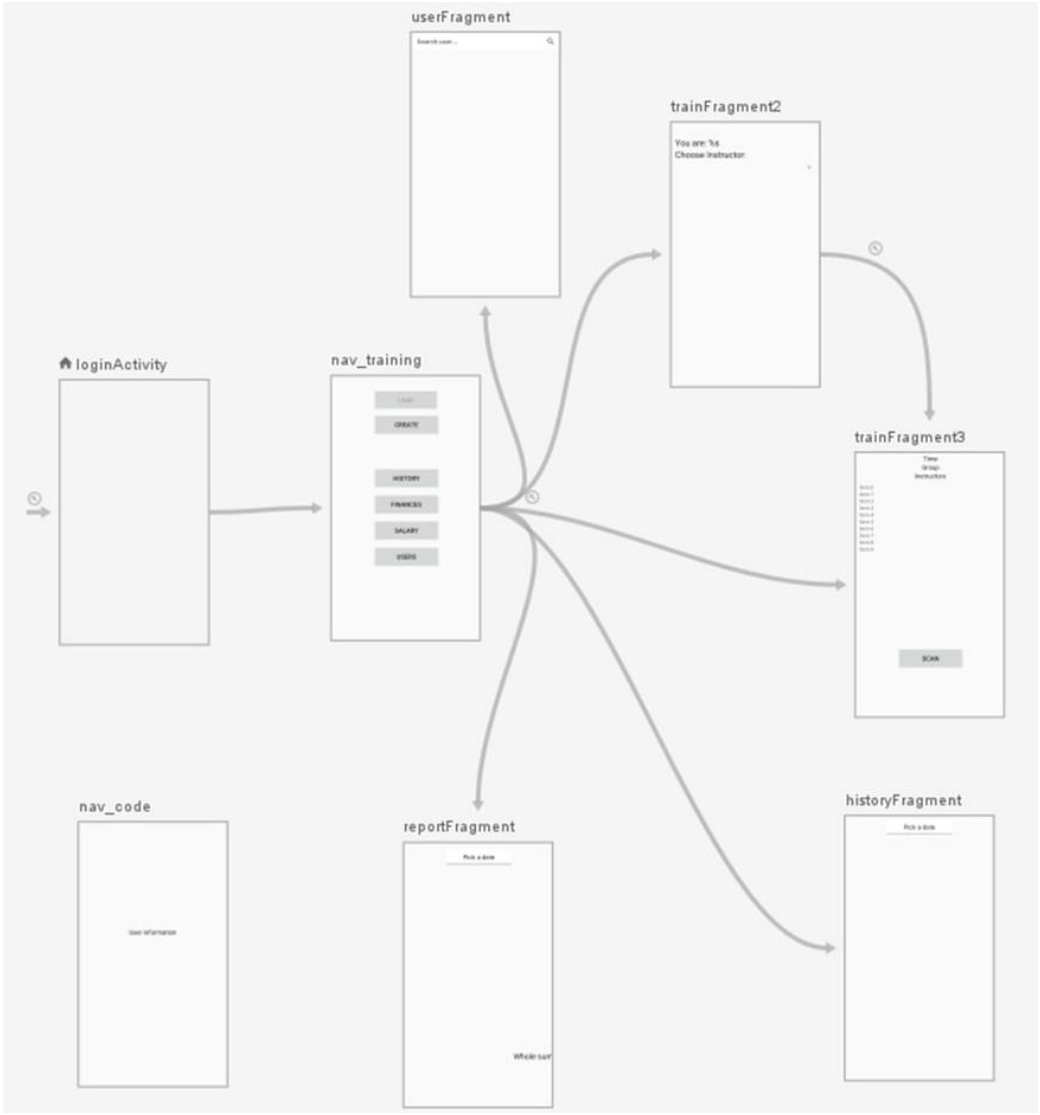


Рисунок 2.2 – Граф навігації мобільного застосунку для студії танців «SDC»

На графі можна побачити схематичне зображення описаних раніше екранів (або фрагментів в контексті навігаційної архітектури) і шляхи можливих переходів. Також в XML-описі графа можна встановити шлях повернення екранів назад по стеку, додаткову анімацію та додаткові значення, що можна передати з фрагмента. Детальний опис інтерфейсу знаходиться у пункті нижче.

Домашнім (тобто першим при запуску) екраном встановлений екран авторизації (loginActivity на графі), оскільки доступ до застосунку повинен бути обмеженим з використанням парольної автентифікації.

Після проходження екрану авторизації наступним відкривається екран домашньої сторінки (nav_training на графі), з якого можна переміститись на усі інші екрани, обираючи відповідну кнопку. При спробі повернення назад – застосунок завершує роботу.

Одним з варіантом переходу з екрану домашньої сторінки є екран створення сесії (trainFragment2), після якого можна перемістится на екран контролю поточної сесії (trainFragment3).

На екран контролю поточної сесії також можна перейти з домашнього екрану, якщо сесія вже є створеною. Кнопка повернення з екрану контролю поточної сесії буде перекидати користувача застосунку на домашній екран.

Окрім цього існують три додаткових екрани, на які можна перейти з домашнього, і які не мають продовження подальшого просування, оскільки є атомарними – виконують весь необхідний функціонал з використанням лише одного екрану. Це екран звітності (reportFragment), екран перегляду історії (historyFragment) і екран учнів (userFragment). При поверненні в стеку для будь-якого з цих екранів – користувача буде переміщено на домашню сторінку.

Внизу головного екрану розміщене нижнє меню, яке дозволяє з будь-якого екрану перейти на домашній екран або на екран сканування коду учня.

Код реалізації навігаційної архітектури знаходиться в Додатку Б.

2.3 Опис програмних рішень

Оскільки реалізація програмних рішень застосунку тісно пов’язана з сервером, доцільно описати структуру сервера, типа запитів, що повертаються, функціонал та API-взаємодію [3].

2.3.1 Опис програмних рішень на стороні сервера

Створення сервера проводилось з використанням мови програмування Python та веб-фреймворку Flask [5]. Створений сервер має широкий функціонал і його найголовнішим модулем є взаємодія з мобільним застосунком, а саме обробка дій, що надходять з додатку, та запис результатів на рівень даних – в базу даних. Код реалізованого клієнтського модуля на стороні сервера знаходиться в Додатку А.

Для обробки кожного з надісланих на сервер запитів були створені серверні функції, які у якості параметрів адресу та HTTP-метод, за яким до них звертаються. При отриманні запиту фреймворк зчитує адресу і запускає відповідну системну функцію, яка буде обробляти запит та повертати певний результат. Приклад такої функції знаходиться в лістингу 2.1.

Лістинг 2.1 – Приклад серверної реалізації однієї з функцій обробки

```
@bp.route("/client/get_users", methods=["POST"])
@init_client
def get_users():
    db = get_db()
    cursor = db.cursor(dictionary=True)

    cursor.execute("SELECT user_name FROM users")
    clients = cursor.fetchall()
    all_names = [client["user_name"] for client in clients if
client['user_name'] != 'Unknown']
    all_names.sort()
    cursor.close()
    return jsonify(message="Send", all_names=all_names)
```

Об’єкт bp – виступає об’єктом реалізованого модуля фреймворку, його метод route(link) створює певний «серверний-хук», який буде активовувати відповідну серверну функцію при переході за посиланням, що вказаний як перший параметр. Серверна функція повертає результат обробки, використовуючи функцію jsonify(), що повертає передані в неї аргументи клієнту у форматі JSON.

Загалом серверний модуль можна схематично розділити на декілька функціональних блоків:

– блок авторизації;
– блок контролю сесії;
– інформаційний блок;
– блок звітності.

Блок авторизації проводить перевірку надання доступу до функціоналу системи. Він складається з трьох серверних функцій – register_token(), яка отримує дані для авторизації, порівнює хеші з збереженими в базі, і створює тимчасовий токен доступу для цього клієнту; token_delete(), яка видаляє надісланий токен доступу; init_client(func) – спеціальна функція, створена по шаблону Декоратора, яка обгортає більшість серверних функцій, і виконує перевірку наявності токена в серверному запиті та в словнику активних токенів. Якщо поданий токен присутній в словнику – декоратор дозволяє подальшу роботу з серверною функцією, інакше – повертається інформація про помилку.

Блок контролю сесії надає функціонал для роботи з сесією – створення, перегляд активної, додання чи видалення учнів з неї. Він складається з наступних функцій:

– get_lists() – отримує з бази даних список груп та інструкторів і надсилає їх у відповідь.

– create_lesson() – отримує від клієнта групу та другого інструктора, після чого створює сесію, яку зберігає в спеціальному об’єкті Lesson. У

випадку якщо інструктор, що надіслав запит, вже має активну сесію – закриває її та створює нову. Повертає інформацію про створену сесію.

– close_lesson() – серверна функція, яка викликає функцію submit_lesson для закриття сесії. Повертає інформацію про стан закриття сесії.

– submit_lesson() – записує інформацію про тренування у таблицю records для усіх учнів та оновляє їх абонементи. Після завершення збереження даних видаляє сесію.

– get_lesson() – повертає для інструктора, що надіслав запит, всю інформацію про поточну сесію з об’єкту Lesson, якщо така існує.

– add_existing(code, instr_id) – функція, що приймає на вхід значення QR коду учня та ідентифікатор інструктора, який ініціював запит. Реалізує логіку запису одного учня декілька раз у тренування, додатково перевіряючи можливість такого запису згідно кількості допустимих тренувань. Якщо інструктор спробує додати учня більше раз, ніж допускають поточні абонементи – буде надіслана пропозиція про створення нового абонементу. Повертає результат процесу запису або додаткову інформацію, необхідно для прийняття подальших рішень.

– adder(code, instr_id) – функція, що приймає на вхід значення QR коду учня та ідентифікатор інструктора, який ініціював запит. Якщо учень вже присутній у сесії – викликає функцію add_existing() з аналогічними параметрами і делегую перевірку правил їй. В іншому випадку перевіряє наявність абонементу і кількість допустимих тренувань на ньому. Якщо все гаразд – записує клієнта до переліку учнів в об’єкті Lesson. Повертає результат процесу запису або додаткову інформацію, необхідно для прийняття подальших рішень.

– add_client() – серверна функція, яка викликає функцію adder() для додання учня. Повертає результати додання учня.

– create_user() – реалізує процес створення учня, використовуючи отриману інформацію. Повертає результати створення.

– create_abon() – реалізує процес створення абонементу для учня. Повертає результати створення.

– delete() – проводить видалення учня з списку учнів для активної сесії за отриманим кодом від клієнта. Повертає результати видалення.

– get_users() – надсилає у відповідь список імен всіх учнів для реалізації механізму додання учня без сканування коду.

– get_code() – надсилає у відповідь код, який прив’язаний до наданого в запиті імені учня, та додаткову інформацію про абонементи.

– find_code() – знаходить код, що прив’язаний до наданого в запиті імені учня, після чого передає його в функцію adder() для додання учня до поточної сесії. Повертає результати додання учня.

– add_simple_abon() – реалізує процес додання одноразового абонементу, що не прив’язаний до учня, до активної сесії. Повертає результат додання.

Інформаційний блок зчитує інформацію про учнів та надсилає її до мобільного застосунку. Складається з таких функцій:

– get_info() – отримує від клієнта код учня, виконує пошук усієї доступної інформацію (персональної та про абонементи) для учня, за яким закріплений просканований код. Якщо код ще не є прив’язаним до певного учня або немає активного абонементу – відсилає відповідь з додатковою інформацією.

– get_all_users_info() – повертає зчитану з бази даних інформацію про всіх клієнтів в поєднанні з інформацією про активні абонементи клієнтів.

– change_user_code() – міняє код учня на новий, наданий у запиті. Повертає результат зміни.

– get_session_hist() – повертає інформацію про проведені тренування за активним абонементом учня, код якого був отриманий з запиту.

– update_user() – оновлює інформацію про користувача згідно надісланих у запиті даних. Повертає результати оновлення.

– get_name() – повертає ім’я користувача з його номером картки за отриманим кодом.

Блок звітності надає функціонал по перегляду проведених інструктором тренувань, інформації про зарплату та фінанси. Містить наступні серверні функції:

– get_lessons_dates() – повертає перелік дат проведених інструктором тренувань.

– form_lessons(old_lesson, i_id) – функція, яка приймає на вхід список інформації про тренування з БД, та ідентифікатор інструктора. Переводить список рядків з тренуваннями у формат придатний для читання мобільним застосунком. Додатково рахує фінанси (суму куплених абонементів) за тренування. Повертає всю зібрану інформацію.

– get_history – виконує пошук тренувань за наданою датою. Якщо інструктором було проведене лише одне тренування або він додатково вказав, інформацію з якого тренування він хоче отримати – викликає функцію form_lessons() і повертає її результати. В іншому випадку повертає список усіх проведених тренувань для подальших дій з сторони застосунку.

– get_reports() – повертає список фінансів за наданий інструктором місяць.

– get_salaries() – повертає розраховану зарплату за наданий інструктором місяць, відповідно від кількості учнів для кожного з тренувань, коефіцієнту нарахування за дати та коефіцієнт за проведення тренування без другого інструктора.

Також створений спеціальний клас Lessons, який реалізує логіку збереження інформації про декілька сесій, та спільний доступ для двох інструкторів.

2.3.2 Опис програмних рішень на стороні мобільного застосунку

Оскільки в попередній частині роботи був проведений аналіз мови програмування і було обрано МП Kotlin, розробка мобільного застосунку для студії танців «SDC» проводилась з використанням Kotlin.

Кожен Android-застосунок складається як мінімум з трьох типів файлів – програмної логіки, ресурсів та конфігурацій [4].

Основними конфігураційними файлами застосунку виступають – AndroidManifest.xml, в якому зберігається назва пакунку та області назв, перелік компонентів застосунку, дозволів та інших особливостей, та gradle-файли, що зберігають інформацію про побудову застосунку – мінімальну та рекомендовану версію SDK, версія побудови, версія та підключення додаткових бібліотек та плагінів.

Файли ресурсів містять інформацію, яку може використати розробник в процесі створення додатку – список речень в залежності від мови, якщо підтримується багатомовність, файли стилів, файли опису інтерфейсу, векторні файли іконок та багато інших можливих варіантів.

Усі файли, що були розроблені, знаходяться в Додатку Б.

Оскільки мобільний застосунок є багатоекранним була необхідність у створенні окремої сукупності класів, що реалізує програмну логіку:

1. MainActivity – головний клас, з запуску якого розпочинається робота застосунку, оскільки він декларований у файлі маніфесту як лаунчер. Він запускає основний вигляд додатку – вікно, в якому міститься контейнер для фрагментів навігаційної архітектури та нижня панель. Саме тут ініціюється контролер навігації (метод setupNav()), з використання якого в подальшому буде проводитись перехід між фрагментами. Також у методі onBackPressed() переписується логіка роботи виходу з додатку, яке буде відбуватися на двох основних екранах – домашньому та сканування коду.

Також тут розміщений метод створення клієнту веб-запитів, що буде використовувати власноруч підписаний SSL-сертифікат.

2. LoginActivity – перший фрагмент, який буде відкривати навігаційний контролер. В ньому описана логіка процесу авторизації і під’єднується вікно авторизації.

Цей фрагмент використовує MVC-підхід до архітектури – існує файл моделі, що буде описаний нижче (LoginViewModel), файл вигляду і файл контролера (LoginActivity). Цей контролер створює об’єкт Наглядач, що спостерігає за станом елементів вводу. При коректному заповнені полів логіну та паролю – він активує кнопку, що надсилає запит на створення точену доступу. При успіху – він записує токен в компаньйон об’єкт MainActivity, що робить його доступним для всіх інших класів. Також він запам’ятовує введений логін в керований файл налаштувань додатку, що дозволить не вводити логін ще раз при наступному запуску застосунку для студії танців «SDC» .

3. LoginViewModel – модель для опису допустимих дій над екраном авторизації. В ній описані методи, які перевіряють довжину введених значень, і попереджають, якщо умови не виконуються. А також метод login(phone, password), який описує логіку запиту реєстрації токену.

4. ClientHTTP – особливий клас, який не описує логіку роботи екранів, проте виконує важливу роль в застосунку. Він надає інструментарій для проведення веб-запитів з сервером.

Метод init викликається при ініціалізації об’єкту і він створює об’єкт клієнта (MainActivity.getSelfCertClient()), що використовує власний SSL-сертифікат.

Метод createCallback(func) являє собою реалізацію шаблону Декоратор, і він приймає параметром функцію, яка буде обробляти отримані від сервера дані. У випадку провалу він виводить повідомлення про помилку. При

успішному отриманні відповіді – метод виконує демаршалінг і перетворює дані в об’єкт JSON-у.

Метод sendRequest(url, parameters, callback) виконує процес створення та надіслання запиту.

5. TrainFragment1 – фрагмент, що реалізує логіку домашньої сторінки після процесу авторизації.

При створенні (метод onCreateView()) він надсилає на сервер запит про перевірку наявності активної сесії для авторизованого інструктора, і якщо така наявна – активує кнопку перегляду активної сесії.

Логіка прив’язки функціоналу до інших кнопок описана в методі onViewCreated(). Кожна з кнопок веде до відповідного фрагменту графу навігації.

6. CustomScannerActivity – активність, яка описує логіку вікна сканування QR коду. Єдиний файл реалізований з допомогою використання мови програмування Java.

7. CodeFragment – фрагмент, що реалізує логіку роботи екрану сканування коду.

При створенні (метод onCreateView()) він створює екземпляр CustomScannerActivity з підключенням бібліотеки zxing для перетворення QR коду в цифровий вивід, і запускає сканер.

В методі onActivityResult() описується результат виконання роботи сканера. Якщо код проходить перевірку – надсилається веб-запит на отримання інформації по цьому коді. Від сервера можуть прийти наступні варіанти відповіді: «Учня з таким кодом не існує» – викликається метод createUserDialog(), який будує діалог для реєстрації учня за цим кодом, новий запит відправляється на сервер; «Абонементу не існує» або «Інформація» – надана інформація по учню відображається на екрані, активуються кнопки створення абонементу (запускає метод createAbonDialog()) та перегляду історії тренувань (запускає метод getAbonHistory()).

8. TrainFragment2 – фрагмент, що реалізує логіку для екрану створення сесії.

При створенні (методі onCreate()) надсилає запит на сервер для отримання списку груп та інструкторів.

Метод onCreateView() виконує прив’язку отриманої інформації до полів вибору та реалізує логіку поступового з’явлення полів після вибору. А також активує кнопку створення сесії.

При нажаті на кнопку активується метод getMore(group, instructor2), який надсилає запит на створення сесії і виконує перехід до екрану активної сесії.

Також в полі вибору групи реалізований функціонал відображення спершу основних груп інструктора, а при нажаті на кнопку More – відкривається розширений список з груп.

9. TrainFragment3 – фрагмент, що реалізує логіку для екрану контролю активної сесії.

При створенні (методі onCreateView()) зчитує передані аргументи – список клієнтів та їх абонементів у відповідні змінні, та активує допоміжне меню вгорі зліва.

Методи onCreateOptionsMenu() та onOptionsItemSelected() описують вигляд та дії, що повинні виконатися при виборі пунктів. Якщо був обраний пункт Закриття сесії – надсилається відповідний запит на сервер і контролер навігації повертається назад по стеку фрагментів. Якщо був обраний пункт Вибір учня – надсилається запит на отримання списку учнів та активується метод createAllUsersDialog(), який будує діалог вибору учня для додання його в сесію без сканування коду; також цей метод прописує логіку фільтрації імен. Якщо був обраний пункт Одиничний абонемент – надсилається запит на додання в сесію одноразового абонементу.

В методі onViewCreated() описується прив’язка всіх візуальних складових до змінних, також прив’язується активація сканера

CustomScannerActivity до кнопки Сканування, і можливість видалення учнів з списку, використовуючи жест протягування (ініціалізується клас SwipeToDeleteCallback).

Метод onActivityResult() аналогічний до однойменного методу класу CodeFragment з єдиною відмінністю в додаванні учня в список при успішній відповіді від сервера після надсилання запиту на додавання.

Окрім цього, для методу createAbonDialog() додається виклик методу abonAlert(), діалог якого ще раз запитує чи справді для певного учня треба створити такий абонемент.

Також перед процесом сканування виконується перевірка на присутність учня в списку (метод abonPresence()), який створює діалог, що уточнює правильність дії додавання вже існуючого учня до списку.

10. SwipeToDeleteCallback – клас, що описує процес та логіку жесту пересування елементу для контейнера.

11. HistoryFragment – фрагмент, що реалізує логіку роботи для екрану перегляду історії тренувань.

При створенні (метод onCreateView()) надсилається запит на отримання дат тренувань для інструктора, після чого ці дати передаються в компонент Календар, що робить доступним для вибору лише вказані дати. Після вибору дати – вона надсилається на сервер і отримана інформація відображається на екрані. Якщо в той день було більше одного тренування – створюється діалог з назв груп, в яких було проведене тренування, і формується новий запит для цієї конкретної групи.

12. ReportFragment – фрагмент, що реалізує логіку роботи для екрану перегляду звітності як зарплати, так і фінансів – функціонал однаковий, відмінність лише у адресі.

При створенні (метод onViewCreated()) до кнопки вибору дати прив’язується надсилання запиту на отримання інформації по вибраній даті, яка після цього відображається на екрані.

13. UserFragment – фрагмент, що реалізує логіку роботи для екрану перегляду всіх користувачів.

При створенні (метод onCreate()) надсилається запит на отримання списку усіх учнів та короткої інформації про них.

В методі onCreateView() заповнюється контейнер учнів, прописується логіка для поля пошуку за іменем та прив’язується створення діалогу з допустимих дій над учнем до вибору елементу з списку.

2.4 Опис інтерфейсу мобільного застосунку для студії танців «SDC»

При запуску мобільного застосунку з’являється екран авторизації показаний на рисунку 2.3.

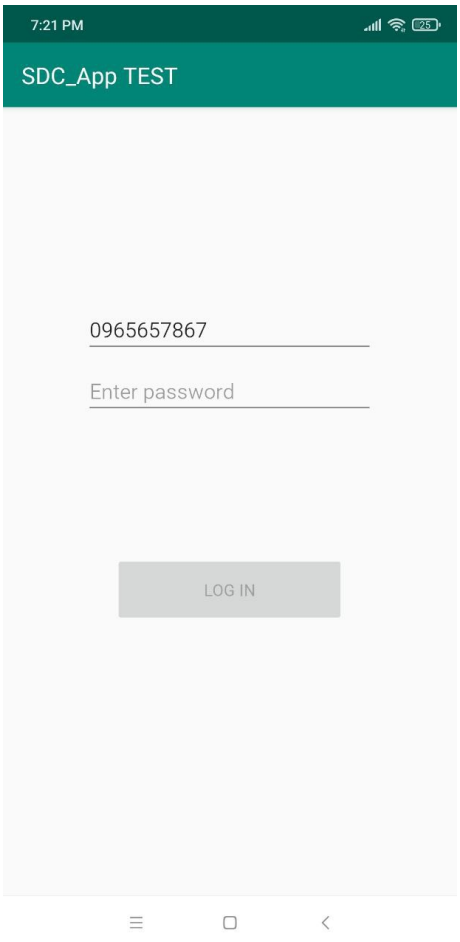


Рисунок 2.3 – Екран авторизації

Якщо в минулому авторизація пройшла успішно – в полі логіну буде зберігатися логін минулого входу. Кнопка авторизації є неактивною, оскільки не всі поля заповнені.

Одразу після входу користувача застосунку переносить на домашній екран, продемонстрований на рисунку 2.4, з якого буде проводитись подальша навігація.

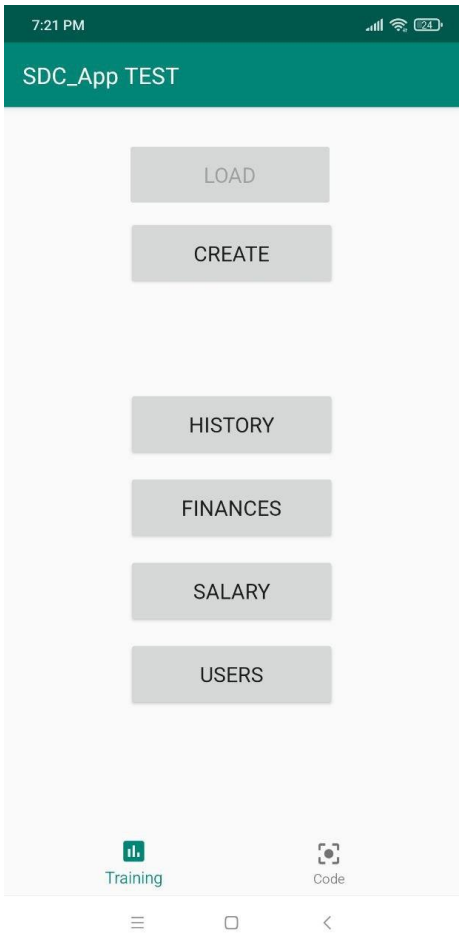
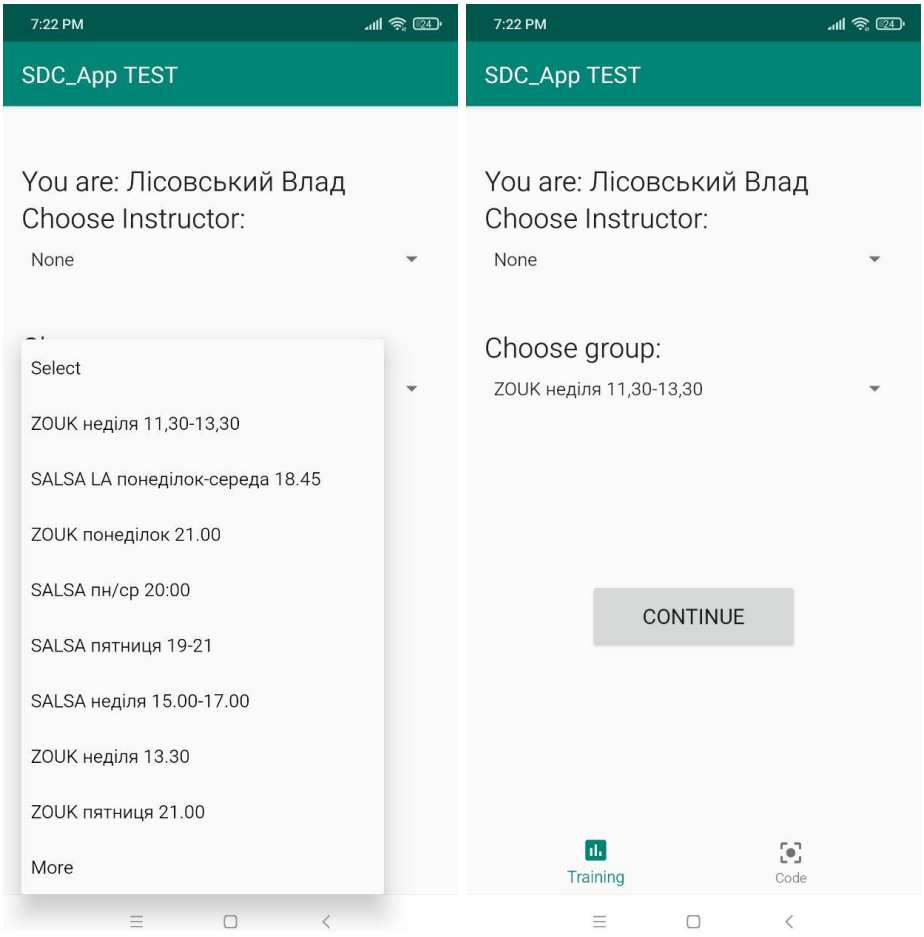


Рисунок 2.4 – Екран головної сторінки

Візуально домашній екран складається з трьох блоків – блок роботи з сесією, блок звітності та нижнє меню переходу між двома головними екранами. Перший блок складається з двох кнопок – Загрузки активної сесії (яка зараз не є доступною, оскільки активна сесія відсутня) і Створення нової

сесії. Другий блок складається з чотирьох кнопок – Історії, Фінанси, Зарплата і Учні. Нижнє меню доступне на всіх екранах.

При натиску на кнопку Створення сесії користувача переносить на екран створення нової сесії, зображений на рисунку 2.5.



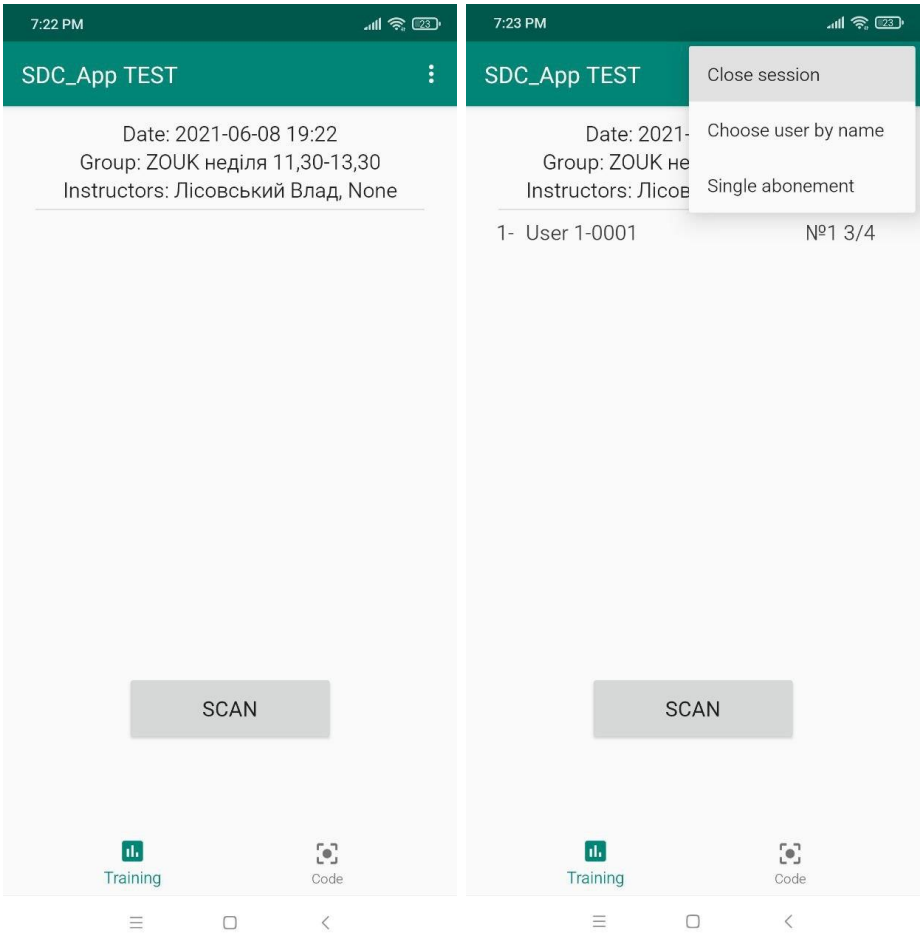
а) б)

Рисунок 2.5 – Екран створення сесії (а – з відкритим полем вибору групи; б – повністю заповнене вікно, готове до продовження)

Екран створення складається з трьох інтерактивних елементів, кожен з яких з’являється після виконання певної умові. Перший створюється зразу після переходу – це поле вибору інструктора, з яким буде проводитись створена сесія. Другий елемент з’явиться після вибору інструктора – це поле вибору групи. Для цього поля існує спеціальний функціонал – спочатку

відображені лише спеціалізовані групи, до яких прив’язані конкретні інструктори, а при натиску на пункт Більше – список вибору перезапуститься, і цього разу буде містити інформацію про всі валідні групи. Третій елемент – це кнопка Продовження, яка буде надсилати інформацію на сервер, і створювати тренування. Вона з’явиться лише після вибору в всіх полях.

Після створення інструктор переходить на екран контролю активної сесії, зображений на рисунку 2.6а. Також цей екран для розширення функціоналу містить додаткове допоміжне меню, що продемонстроване на рисунку 2.6б.



а) б)

Рисунок 2.6 – Екран контролю активної сесії (а – вигляд пустої сторінки; б – вигляд з допоміжним меню)

Основний екран складається з «шапки», в якій розміщена текстова інформація про поточну сесію – дата, назва групи та список інструкторів, з основного блоку, де розміщується контейнер, в якому буде зберігатись інформація про записаних на тренування учнів, та кнопки Сканування коду.

При нажаті на кнопку Сканування відкривається окремий екран сканера, зображений на рисунку 2.7, в якому розміщене поле сканування і кнопка ввімкнення ліхтарика.

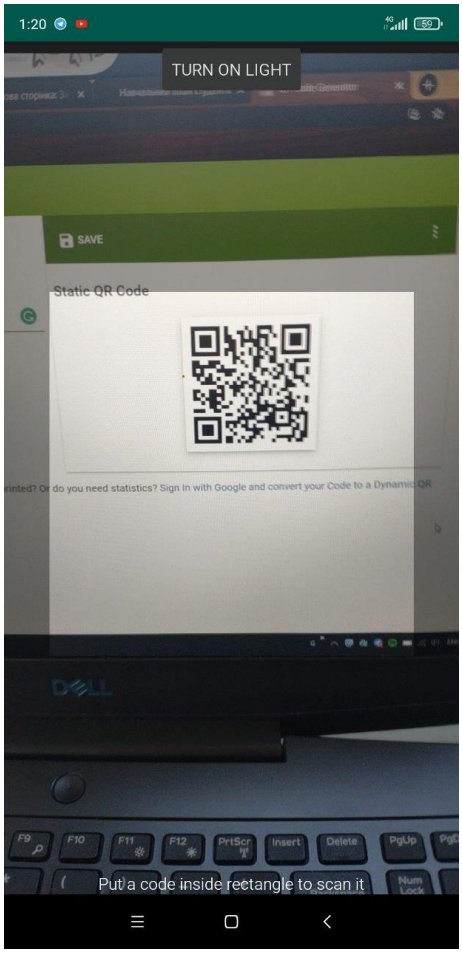
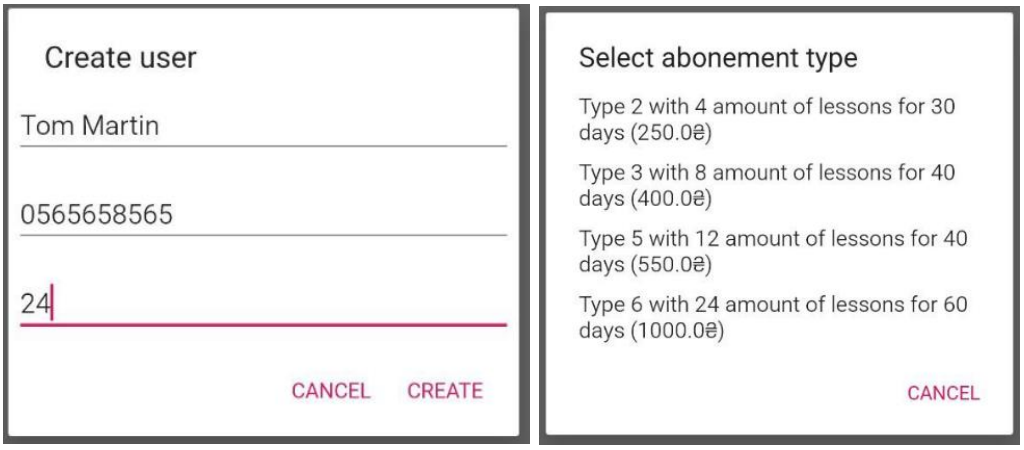


Рисунок 2.7 – Екран сканера

При завершенні сканування просканований код надсилається на сервер і в залежності від відповіді приймаються конкретні дії. Якщо учня з таким кодом не існує в базі даних – відкривається діалог створення учня, продемонстрований на рисунку 2.8а, яке містить три поля – імені, номеру

телефону та номеру картки SDC. Якщо учень з таким кодом існує, але у нього немає активного абонементу – відкривається діалог створення абонементу, зображений на рисунку 2.8б, в якому міститься перелік доступних абонементів з описом кожного типу.



а) б)

Рисунок 2.8 – Діалогові вікна (а – діалог створення користувача; б – діалог створення абонементу)

При виборі бажаного типу абонементу створюється діалог-попередження, показаний на рисунку 2.9, з інформацією про обраний абонемент. При доданні вже існуючого в сесії учня створюється діалог-попередження, що даний учень вже є присутнім в сесії, продемонстрований на рисунку 2.10. Якщо від сервера прийде інформація, що доступних тренувань для нового додання учня не вистачає – буде запущений діалог створення абонементу.

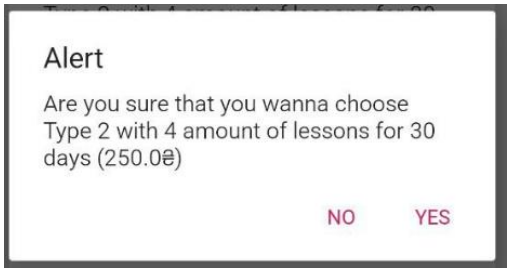


Рисунок 2.9 – Діалог-попередження про вибір абонементу

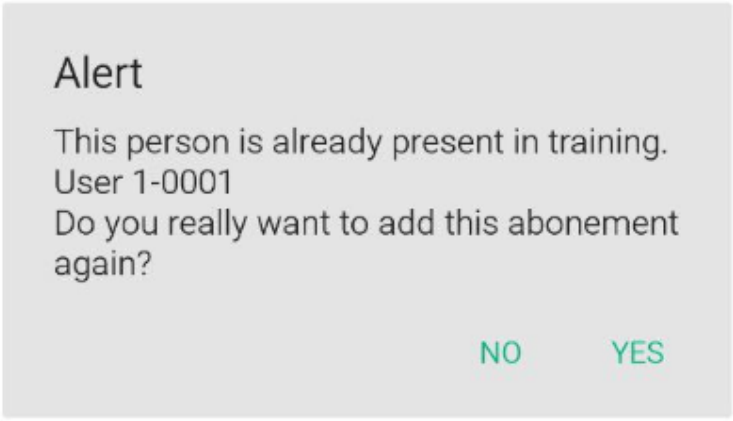
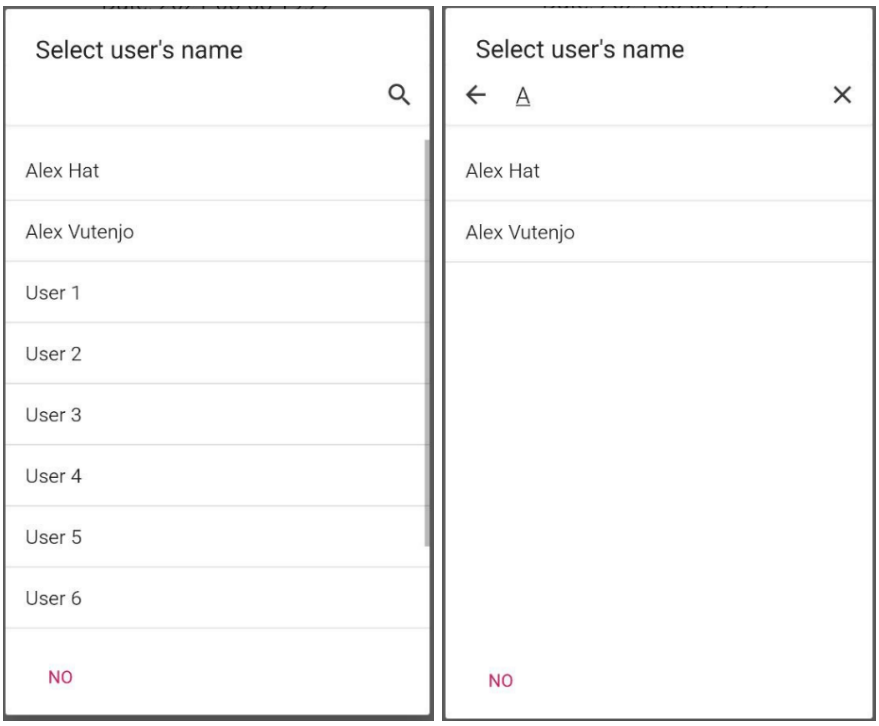


Рисунок 2.10 – Діалог-попередження про додання існуючого в сесії учня

Допоміжне меню на екрані контролю сесії складається з трьох пунктів – закриття сесії, додання учня без сканування коду та додання одноразового анонімного учня. Перший пункт надсилає запит закриття сесії та повертає інструктора на домашній екран. Другий пункт відкриває діалог вибору учня, показаний на рисунку 2.11а, також вгорі діалогу розміщене поле пошуку, яке буде фільтрувати результати пошук, демонстровано на рисунку 2.11б.

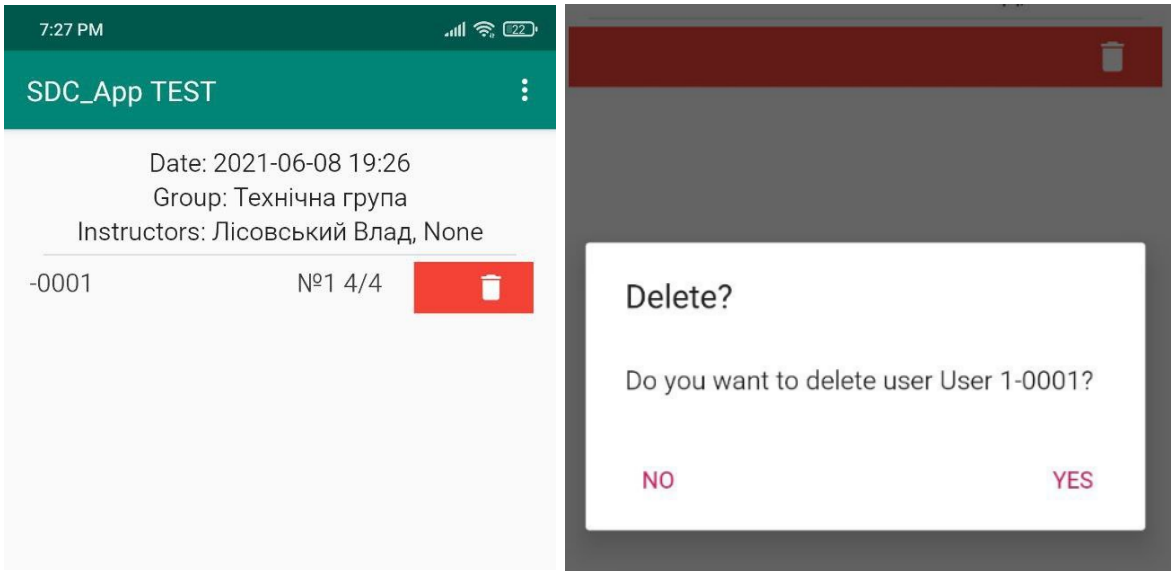


а) б)

Рисунок 2.11 – Діалог додання учня (а – повний список; б – відфільтрований)

Третій пункт допоміжного меню додає одноразового анонімного учня, який не прив’язаний до реальних людей, а існує лише для реалізації пробного тренування.

Останньою функціональною можливістю доступною на екрані контролю сесії є видалення внесеного учня з тренування. Для цього необхідно потягнути край елемента з бажаним учнем справа наліво, демонстрація знаходиться на рисунку 2.12а. Функція видалення викличе діалог-попередження про видалення, показаний на рисунку 2.12б, після підтвердження якої елемент з анімацією пропаде, а нижні – пересунуться вище.



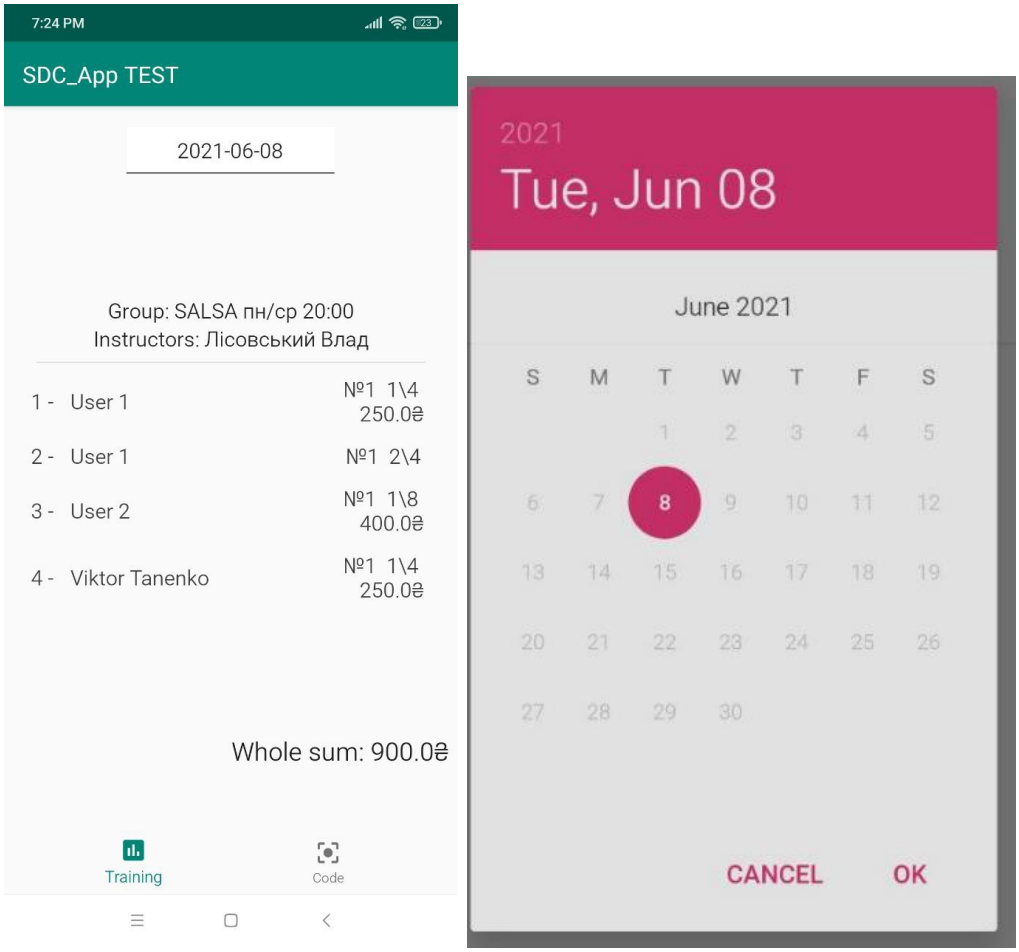
а) б)

Рисунок 2.12 – Функція видалення учня (а – рух протягування елементу; б – діалог-попередження)

При поверненні на домашній екран, коли присутня активна сесія – перша кнопка загрузки сесії буде активною, що дозволить повернутися назад до екрану контролю сесії.

При нажаті на кнопку Історія на домашньому екрані інструктор перейде до екрану перегляду історії, продемонстрований на рисунку 2.13а.

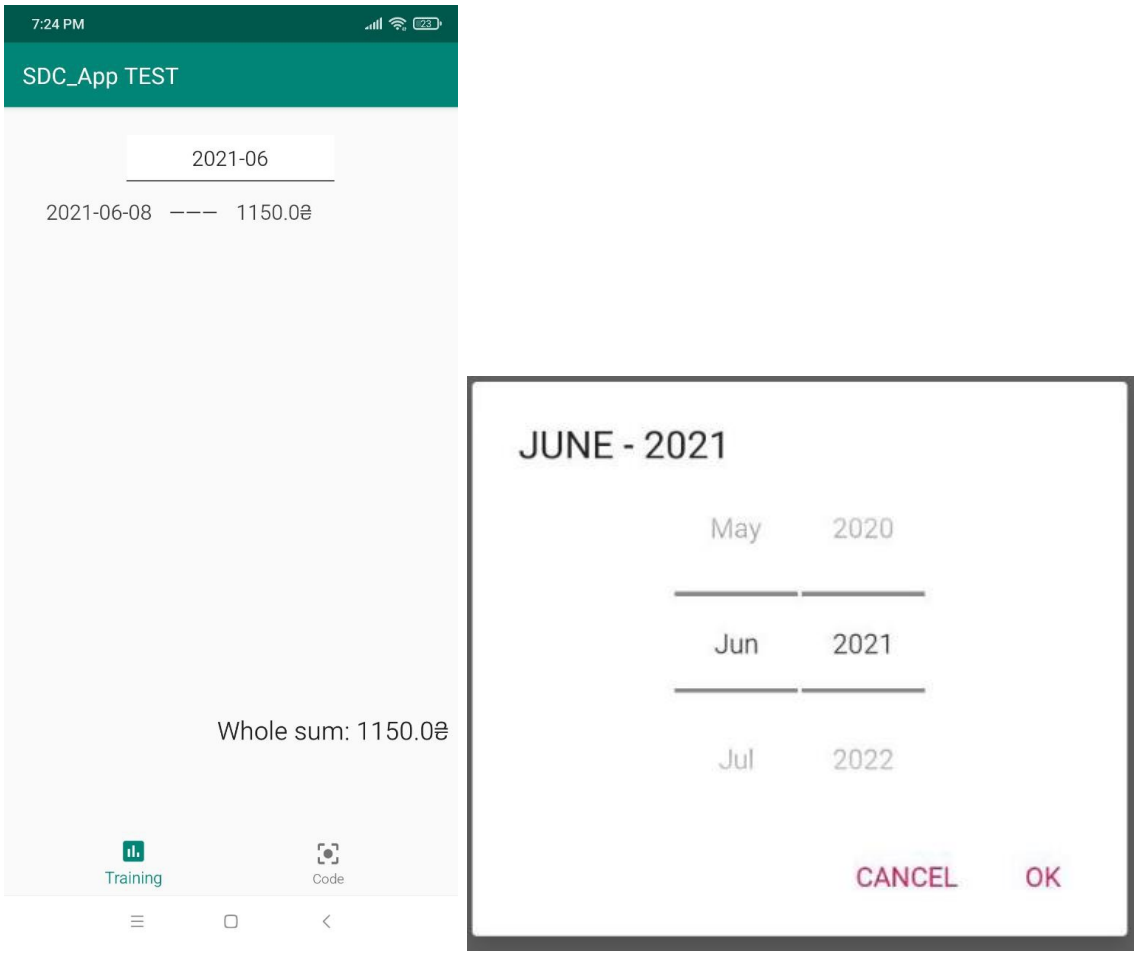
Екран складається з кнопки вибору дати, яка відкриє віджет календар, показаний на рисунку 2.13б, в якому можна обрати дати попередніх тренувань, і блоку з текстовою інформацією – назвою групи, списком інструкторів, списком учнів та вартістю абонементу, якщо він був куплений під час тренування, і загальна сума, що була отримана під час тренування. У випадку, якщо під час обраної дати було проведено декілька тренувань – буде створений діалог з списком збережених тренувань для конкретного інструктора, який дозволить вибрати бажане тренування.



а) б)

Рисунок 2.13 – Екран перегляду історії (а – заповнений екран; б – віджет календар)

При нажаті на кнопку Фінанси на домашньому екрані інструктора перенесе до екрану перегляду звітності по фінансах, зображений на рисунку 2.14а, який складається з кнопки вибору дати, що відкриє діалог вибору бажаного року та місяця, показаний на рисунку 2.14б, і блоку текстової інформації з датами та загальною сумою.



а) б)

Рисунок 2.14 – Екран перегляду фінансів (а – заповнений екран; б – діалог вибору дати)

При нажаті на кнопку Фінанси на домашньому екрані інструктора перенесе до екрану перегляду звітності по зарплаті, показаний на рисунку 2.15, який складовою схожий до екрану перегляду звітності по фінансах. В кнопці обирається дата і отримана з сервера інформація переноситься в

основний блок. Це інформація згрупована по групам з списком дат проведених тренувань, кількості учнів та зарплати, нарахованої за це тренування. І внизу знаходиться текстовий блок, що містить інформацію про зарплату нараховану за місяць.

Рисунок 2.15 – Екран перегляду зарплати

При нажаті на кнопку Учні на домашньому екрані інструктор перейде до екрану перегляду учнів, продемонстрований на рисунку 2.16. Цей екран складається з поля пошуку, яке буде фільтрувати список, та власне списку учнів. В списку міститься персональна інформація про учня – його ім’я, номер телефону та номер картки SDC, а також інформація про останній

активний абонемент – номер, кількість доступних занять і дата його закінчення.

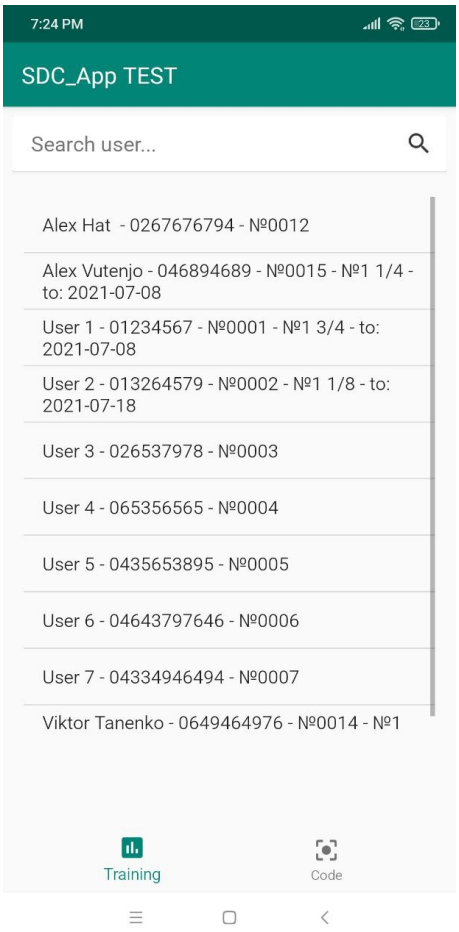
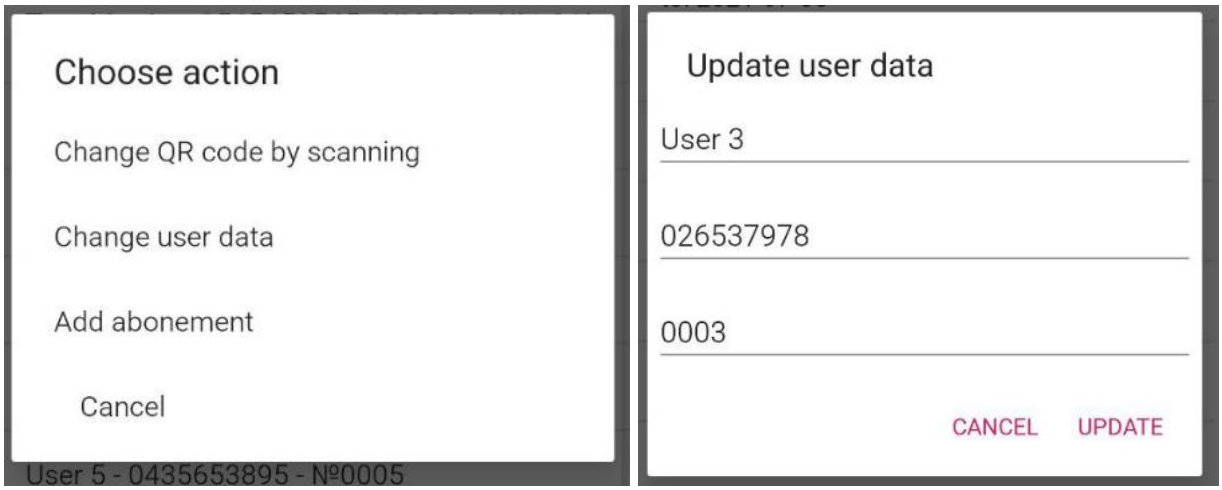


Рисунок 2.16 – Екран перегляду всіх учнів

При утриманні елементу списку (учня) з’являється діалог додаткових дій, що можна провести над цим учнем, показаний на рисунку 2.17а. Всього можливі три дії – зміна QR коду учня, зміна особистих даних і створення абонементу. При виборі першого пункту відкриється екран сканеру (його вигляд був продемонстрований та описаний вище), результат сканування якого відправиться на сервер, і перезапише QR код учня на новий. При виборі другого пункту створиться діалог зміни особистих даних, продемонстрований на рисунку 2.17б, що складається з трьох полів, аналогічних полів реєстрації учня, але цього разу з заповненими даними. При

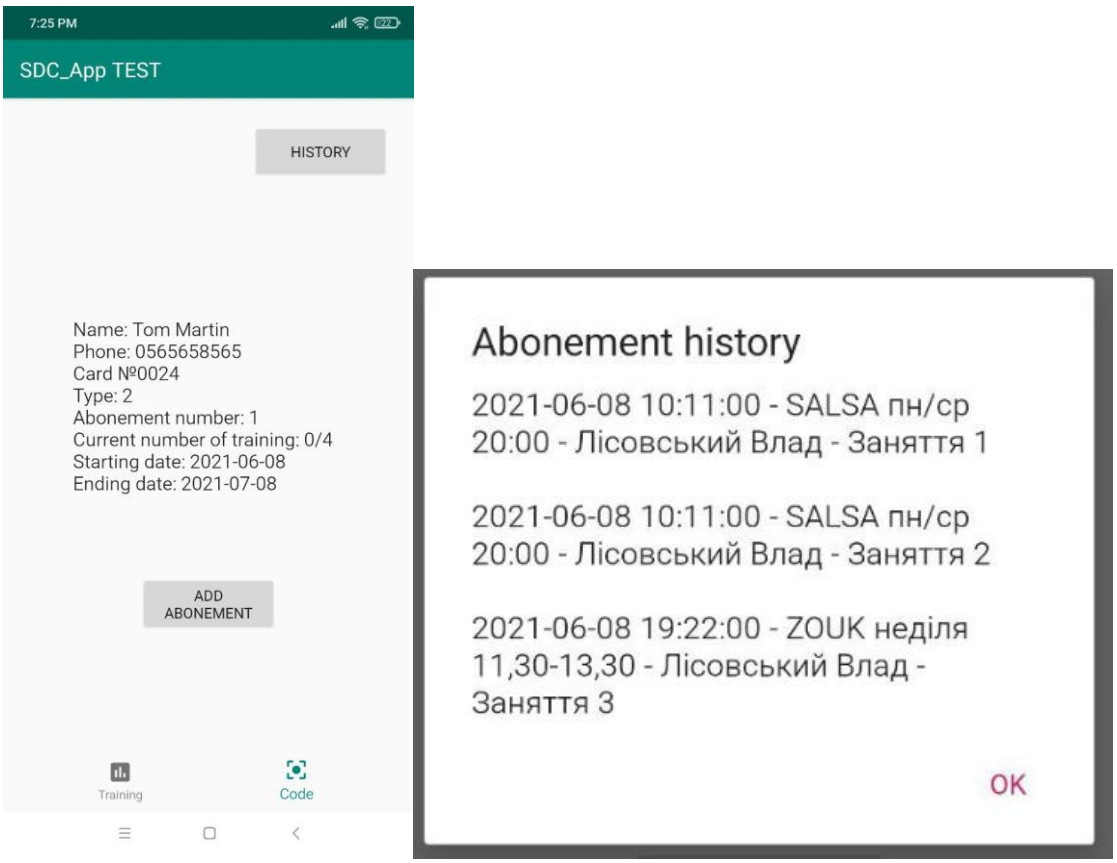
зміні даних – вони відправляються на сервер і перезаписують вже існуючу інформацію. При виборі третього пункту відкриється діалог створення нового абонементу, після вибору якого з’явиться діалог-попередження про створення, тобто аналогічно до діалогу створення абонементу, що був описаний вище.



а) б)

Рисунок 2.17 – Додаткові дії над учнем (а – діалог переліку дій; б – діалог зміни особистих даних)

І останнім являється екран перегляду детальної інформації про учня, на який можна перейти, використавши кнопку Код з нижньої панелі меню. При переході на екран зразу запуститься екран сканера коду, послідовність дій якого схожа до запуску сканера з екрану контролю активної сесії. Єдина відмінність полягає в тому, що в кінці отримана інформація виводить на екран, показаний на рисунку 2.18а. Окрім цього внизу присутня кнопка додання абонементу для учня, що викликає аналогічні до попередніх діалоги створення абонементу та попередження про вибір. Останнім елементом виступає кнопка Історії, яка дозволяє переглянути історію по останньому абонементу, приклад зображений на рисунку 2.18б, тобто перелік занять, на яких був активований абонемент.



а) б)

Рисунок 2.18 – Екран перегляду інформації по учню (а – заповнений екран; б – діалог історії)

При нажаті на кнопку повернення, якщо поточним є екран домашньої сторінки чи екран перегляду інформації про учня – відбувається закриття додатку. У всіх інших випадках відбувається перехід до екрану в стеку вище.

2.5 Тестування та валідація мобільного застосунку для студії танців «SDC»

Для валідації коду мобільного застосунку для студії танців «SDC» було використане інтегроване середовище розробки Android Studio, яке рекомендується для розробки мобільних застосунків, і було обране в першому розділі. Запуск валідації був проведений з використанням утиліти Інспекції коду [7], показаний на рисунку 2.19.

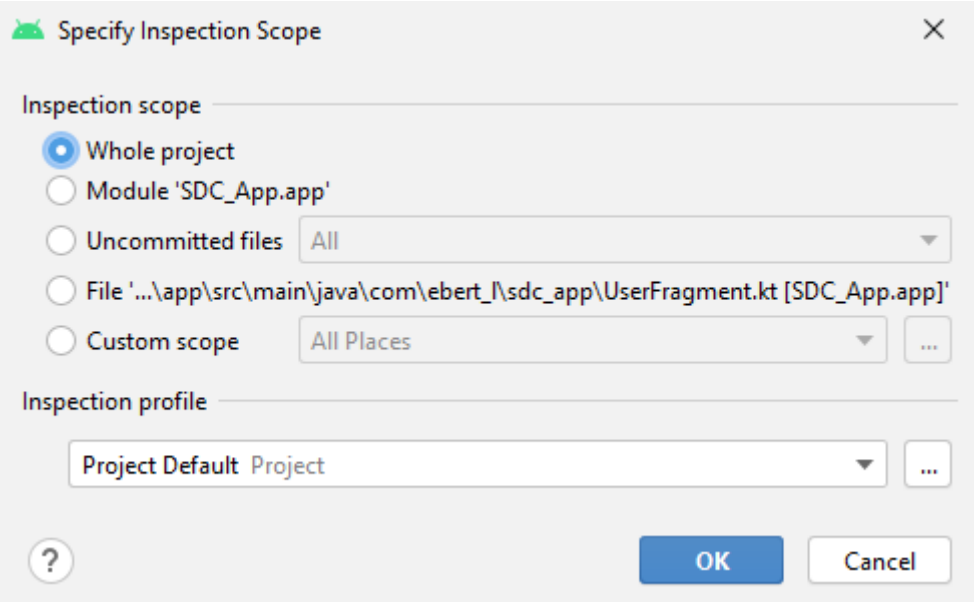


Рисунок 2.19 – Запуски утиліти Інспекції

Всього було виявлено помилки, що належать чотирьом групам – Android, Java, Kotlin та Proofreading. Остання група містить інформацію про типографічні помилки, зображений на рисунку 2.20, майже всі випадки пов’язані з словом «абонемент», якого не існує в англійській мові, що викликає відповідні попередження про незрозумілі назви змінних. Відповідно цю групу можна пропустити.

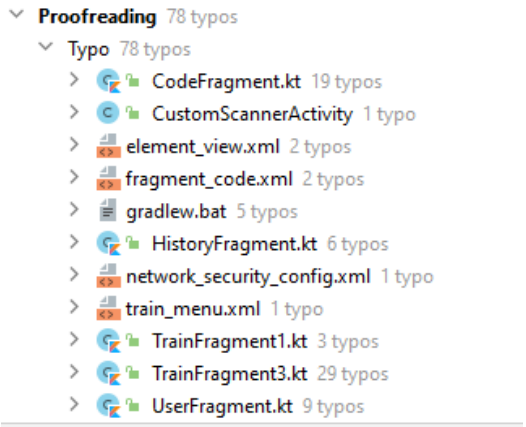


Рисунок 2.20 – Категорія попереджень Proofreading

Перша група, показана на рисунку 2.21, містить попередження для типових Android файлів – конфігурацій та ресурсів.

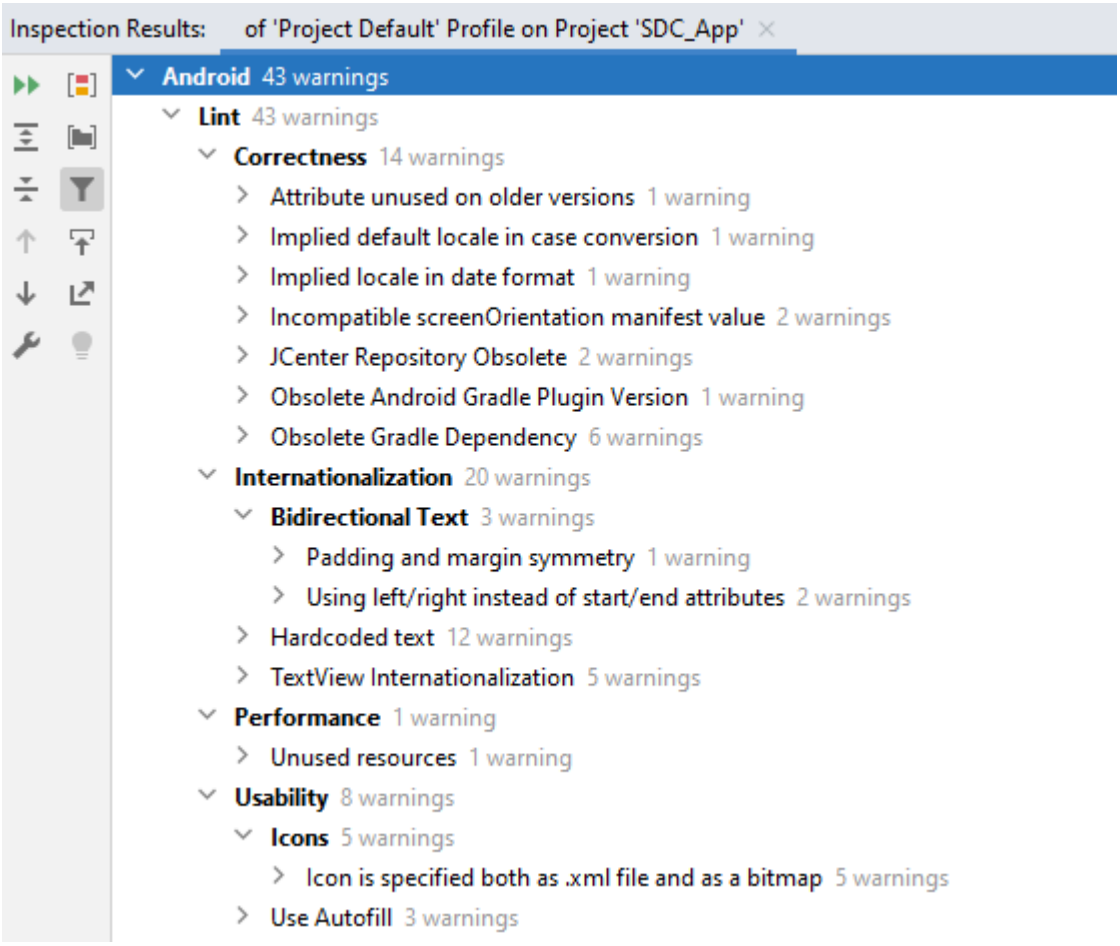


Рисунок 2.21 – Категорія попереджень Android

Був проведений детальний розгляд попереджень:

– «Атрибут не використовується в старіших версіях» – цільова версія SDK – це API 23, а атрибут network_security_config – з’являється з API 24, проте цей файл є необхідним для переліку доменів, з якими працює мобільний застосунок, тому це попередження пропускається;

– «Припускається використання локалі за замовчування для зміни регістру» – приймається запропоноване вирішення цього попередження;

– «Припускається використання локалі за замовчування для формату даних» – пропускається, оскільки формат встановлений на сервері та не залежить від локації;

– «Несумісна орієнтація екрану в значеннях маніфесту» – пропускається, оскільки для роботи необхідна замкнена портретна орієнтація екрану;

– «Репозиторій JCenter застарілий» – виконана заміна на сучасніший репозиторій;

– «Застаріла версія плагіну Android Gradle» – виконана заміна на сучаснішу версію плагіну;

– «Застарілі залежності Gradle» – виконані заміни залежностей;

– «Симетрія відступів» – пропускається, оскільки асиметрія необхідна для елементу, де виникає попередження;

– «Використання атрибутів left/right замість start/end» – виконана заміна;

– «Використання вшитих стрічок» – стрічки перезаписані в ресурси;

– «Інтернаціоналізація TextView» – всі значення TextView, що були вшиті, були перезаписані в ресурси, а конкатенація – замінена на шаблони;

– «Наявні ресурси, що не використовуються» – було проведене видалення таких ресурсів;

– «Використання автозаповнення» – був доданий відповідний атрибут;

– «Іконки» – пропускається.

Наступні дві групи стосуються зауважень щодо синтаксису двох використаних мов програмування – Java та Kotlin, показані на рисунку 2.22.

Попередження для Java були пропущені, оскільки такий синтаксис є необхідним для роботи.

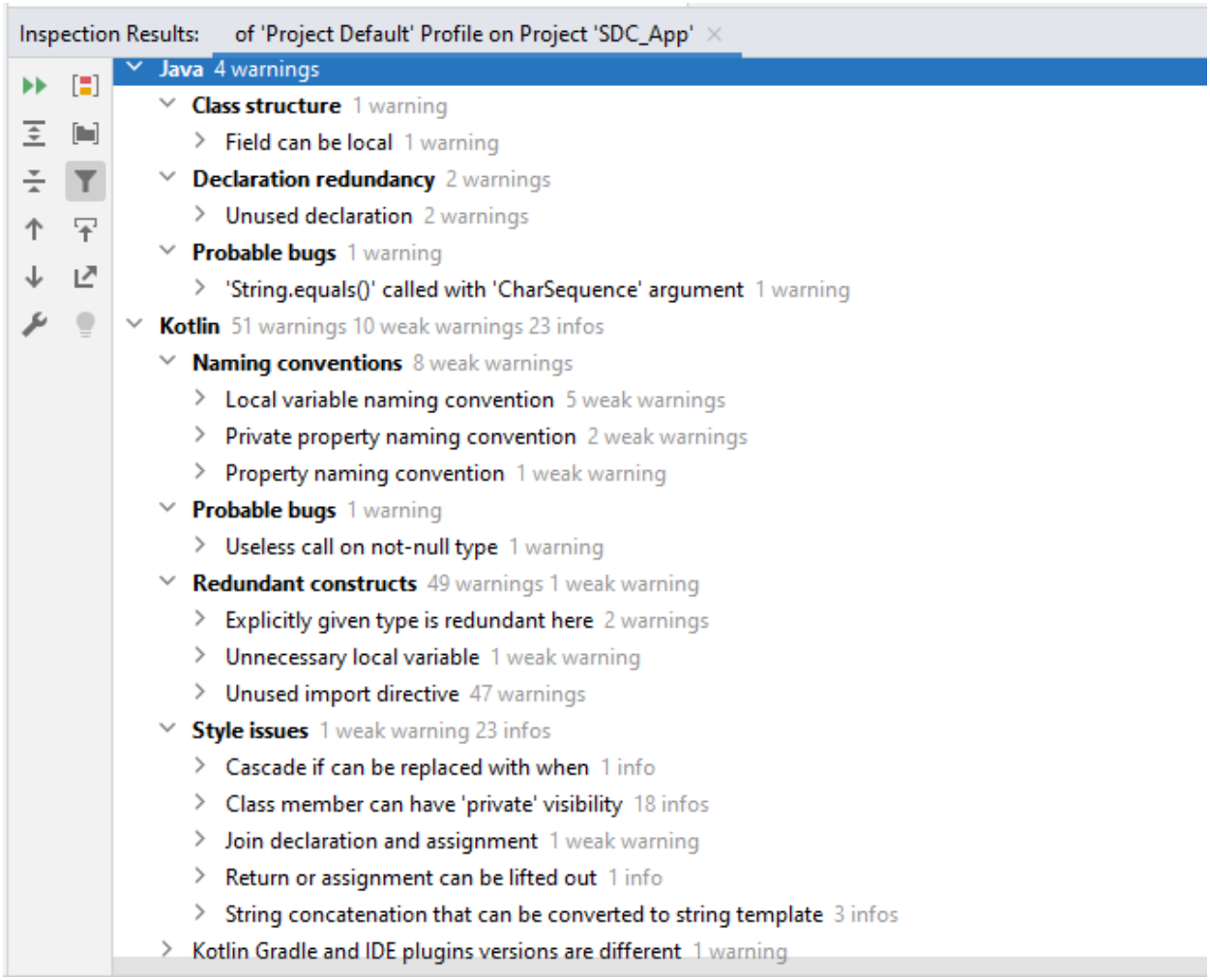


Рисунок 2.22 – Категорії попереджень для мов програмування

Були розглянуті попередження для Kotlin:

– «Недотримання правил для назв локальних змін» – назви змін були виправлені;

– «Недотримання правил для назв приватних атрибутів» – пропущено;

– «Непотрібна перевірка» – перевірка була замінена на простішу;

– «Непотрібне вказання типу» – виправлено;

– «Непотрібна локальна змінна» – створення було переміщено в директиву повернення результату;

– «Непотрібні імпорти» – була проведена оптимізація імпортів, що не використовуються;

– «Каскад if можна замінити на when» – була проведена заміна;

– «Члени класів можуть мати модифікатор приватних» – пропущено;

– «Декларація та присвоєння можуть бути об’єднані» – виконано;

– «Конкатенація стрічок може бути перетворена в форматування стрічок» – виконано.

Процес тестування роботи мобільного застосунку для студії танців «SDC» було проведено з використанням технології чорного ящика. Тобто перевірялось коректне виконання усіх функціональних можливостей інтерфейсу додатку, порівнюючи вхідні та вихідні значення, не звертаючи увагу на прихований процес роботи коду. Аналогічним чином проводилось тестування серверної частини.

2.6 Висновок до другого розділу

В другому розділі кваліфікаційної роботи був описаний процес проектування, розробки та тестування мобільного застосунку для студії танців «SDC». Був проведений аналіз варіантів використання та архітектури застосунку. Був розроблений мобільний застосунок для студії танців «SDC» на основі клієнт-серверної архітектури. Був проведений опис реалізованих програмних рішень як клієнтської, так і серверної частини. Було проведено тестування та валідацію мобільного застосунку для студії танців «SDC».

РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Долікарська допомога при переломах

Перелом – це частковий або повний розрив кісткової тканини, який відбувається під дією сили, що перевищує механічну міцність структури кістки [23]. Перелом, зазвичай, відбувається з пошкодженням прилеглих м'яких тканин – м'язів, жирової тканини, шкіри.

За пошкодженням зовнішніх покривів тіла поділяються на:

– закриті переломи – без пошкодження шкіри;

– відкриті переломи – шкіра пошкоджена в зоні перелому. Відкриті переломи тим небезпечні, що вони можуть інфікувати відламки.

Як закриті, так і відкриті переломи можуть бути зі зміщенням і без нього.

Залежно від ступеня еластичності кісткової тканини переломи бувають повні і неповні. При неповному переломі порушується якась частина поперечних кісток, з'являється тріщина кісток. При повному – кістка розривається на частини.

Характерними ознаками переломів є:

– різкий біль і деформація;

– порушення функції ураженої ділянки;

– набряк і крововилив в зоні перелому;

– патологічна ненормальна рухливість кістки і кісткове хрумтіння;

– нерівність кісток;

– в рані відкритого перелому виступає уламок кістки.

Найважливіше при переломі – це правильно закріпити пошкоджені кістки, суглоби, зв'язані з ними кінцівки в нерухомому і найзручнішому для потерпілого стані.

Долікарська допомога полягає у фіксація кісток у ділянці переломів, протишокових заходах та правильному транспортуванні потерпілого до медичної установи. Правильна фіксація пошкоджених кінцівок попереджує зміщення уламків, зменшує пошкодження судин, нервів, м'язів і шкіри гострими краями уражених кісток, попереджує шок [24].

Для фіксації кісток використовують транспортну шину. Перед накладанням транспортної шини, обробляють перелом йодом і накладають асептичну пов'язку при відкритому переломі.

При транспортуванні шину закріпляють, щоб добре зафіксувати область перелому; під шину вкладають вату, тканину; фіксують два суглоба вище і нижче перелому. Потерпілого утеплюють, дають випити алкоголю, гарячого чаю, кави, вводять морфій.

В залежності від локалізації перелому існують різні загрози, кожна з загроз характеризується своїми ознаками і може вимагати індивідуального підходу для допомоги. Розглядають наступні переломи:

1. Перелом кісток черепа. Ушкодження черепа веде до струсу мозку, стискання, забою черепа. Серед ознак – запаморочення, нудота, блювання, сповільнення пульсу, втрата пам'яті (амнезія), порушення мови і міміки. Переломи виникають при дорожньо-транспортних пригодах в результаті важких ударів в голову, при обвалах, падіннях і супроводжуються стисканням, прогинанням, а потім розривом кісток черепа.

2. Переломи склепіння черепа можуть бути прямі – в місці безпосередньої дії і непрямі – виникають від вторинної дії травмуючого фактора.

Характерні ознаки: витікання спинномозкової рідини і крові з носа, носоглотки, вуха; синці в ділянці очних ямок, які розвиваються через 1-2 доби після травми; порушення функцій черепно-мозкових нервів; локальний біль, деформація черепа, вдавлення, наявність виступів.

Перша допомога: лід на голову, очистити ротову порожнину від блювотиння; на рану накладають асептичну пов'язку, транспортують до лікарні на ношах на спині, під голову кладуть валик чи подушку. Запобігти западанню язика і проходженню блювотних мас.

3. Перелом ключиці. Виникає внаслідок падіння на витягнуту руку, на лікоть чи зовнішню поверхню плеча або при ударі по ключиці.

Характерні ознаки: біль в ділянці перелому ключиці, обмеження рухів, припухлість; підшкірна гематома на місці перелому, деформація, при пальпації відмічається характерне зміщення уламків.

Перша допомога: в пахову ямку кладуть жмут вати, руку згинають під прямим кутом до тулуба, бинтують від кінцівки до спини, підв'язують нижче ліктя косинкою до шиї, на уражене місце накладають холод. Накладають пов'язку Дезо, шину Кузьмінського, використовують вато-марлеві кільця Дельбе.

4. Переломи ребер, плеча. Причини: падіння на виступаючий предмет, наїзд автомобіля, ДТП, удари, стискання грудної клітки в передньо-задньому напрямі при землетрусах, стихійних лихах, аваріях.

Характерні ознаки: сильний біль в місцях перелому; посилення при кашлі, диханні, зміні положення тіла; при пальпації болісно і хрустить в місці перелому.

Перша допомога: накладання тугої пов'язки; стягують бинтами груди під час видиху; транспортують на носилках в напівсидячому положенні.

5. Перелом плечової кістки.

Характерні ознаки: різний біль і патологічна рухомість, іноді крепітація, припухлість; неможливі рухи в плечовому суглобі; при збитому переломі можливі незначні рухи в плечовому суглобі.

Перша допомога: транспортна іммобілізація здійснюється за допомогою шин Крамера, при збитих переломах руку підвішують на пов'язці.

6. Переломи передпліччя. Розрізняють переломи верхньої третини, діалізу і нижньої третини передпліччя.

Характерні ознаки: локалізована болісність; штокоподібна деформація передпліччя; обмеження рухів у ліктьовому суглобі; набряк м'яких тканин у місці перелому; підшкірний крововилив.

Перша допомога: транспортна іммобілізація за допомогою шини Крамера.

7. Перелом кульової кістки. Результат транспортних аварій з характерними ознаками: біль в кульовому суглобі; неможливість стояти, ходити, сидіти; кінцівка злегка повернута всередину, зігнута.

Перша допомога: транспортна іммобілізація на носилках на стороні здорової кінцівки.

8. Перелом кісток гомілки.

Характерні ознаки: біль у колінному суглобі при наступанні; біль у ділянці перелому, припухлість, контури згладжені; вкорочення гомілки; крововилив у колінний суглоб.

Перша допомога: шини Крамера, шини Дітеріхса, спосіб "нога до ноги".

9. Перелом діалізу кісток гомілки. Виникають при прямому ударі, здавленні, падінні на гомілку.

Характерні ознаки: раптовий біль при напруженні розгиначів гомілки; припухлість в ділянці; пасивне положення кінцівки; при переломах обох кісток виявляється рухомість кісток (уламків) і крепітація.

Перша допомога: шина Крамера, накладена від середньої третини стегна до пальців стопи.

10. Перелом, пошкодження щелепи. Перша допомога – потерпілого в сидячому стані транспортують до лікарні з легким нахилом голови вперед; запобігають асфіксії кров'ю, запалим язиком, слиною; накладають фіксуючу пов'язку.

3.2 Методи оцінки соціальної та соціально-економічної ефективності заходів щодо покращенню умов та охорони праці

Ефективність заходів щодо поліпшення умов і охорони праці оцінюється, в першу чергу, за показниками соціальної ефективності, які передбачають створення умов праці, що відповідають санітарним нормам і вимогам правил безпеки. Покращення умов і охорони праці призводить до зменшення кількості виробничих травм, загальної і професійної захворюваності; зменшення кількості випадків виходу на пенсію за інвалідністю внаслідок травматизму чи професійної захворюваності; скорочення плинності кадрів через незадовільні умови праці.

Соціально-економічна ефективність розраховується з метою:

– економічного обґрунтування планових заходів, необхідних для вибору оптимальних варіантів технологічних, ергономічних та організаційних рішень;

– визначення фактичної ефективності заходів щодо поліпшення умов і охорони праці;

– оцінки результатів управління виробництвом на різних рівнях;

– розрахунку необхідних витрат для приведення умов праці на робочих місцях у відповідність до нормативних вимог;

– визначення раціональних розмірів матеріального стимулювання працівників підприємства за розробку і запровадження працеохоронних заходів.

Показники соціальної і соціально-економічної ефективності розраховуються як відношення величин соціальних або соціально-економічних результатів до витрат, необхідних для їх здійснення. Такі показники характеризують кількість умовних одиниць сукупного об'єму соціального чи соціально-економічного результату в розрахунку на одиницю витрат.

Показники соціальної і соціально-економічної ефективності використовуються для визначення фактичного рівня питомих витрат, необхідних для зниження рівня травматизму, захворюваності, плинності кадрів на різних підприємствах та в економіці в цілому.

Економічні аспекти охорони праці оцінюються за допомогою методів оцінки соціальної й економічної ефективності заходів по створенню умов праці, що відповідають чинним нормативним актам з охорони праці [20].

Для оцінки соціальної ефективності заходів з удосконалення умов та охорони праці використовують такі показники:

– скорочення кількості робочих місць, що не відповідають вимогам нормативних актів щодо безпеки праці;

– скорочення чисельності працівників, які працюють в умовах, що не відповідають санітарним нормам;

– збільшення кількості машин, механізмів та виробничих приміщень, приведених до вимог норм охорони праці;

– зменшення коефіцієнта частоти травматизму;

– зменшення коефіцієнта тяжкості травматизму;

– зменшення коефіцієнта частоти професійних захворювань через несприятливі умови праці;

– зменшення коефіцієнта тяжкості захворювання;

– скорочення плинності кадрів через несприятливі умови праці.

3.3 Вимоги до виробничих приміщень для експлуатації ВДТ

Однією із основних характерних особливостей сучасного розвитку суспільства є зростання сфер діяльності людини, в яких використовуються інформаційні технології, де широке розповсюдження отримали ВДТ. Однак їх використання загострило проблему збереження здоров’я працівника, що в

свою чергу вимагає вдосконалення існуючих та розробки нових підходів до організації робочого місця, яке обладнане ВДТ.

Робоче місце – це ділянка, де постійно або тимчасово перебувають працюючі при виконанні ними трудових процесів. Простір, який охоплює всю площу робочого місця і має висоту 2 м над її рівнем, називається робочою зоною [21].

Основні вимоги до виробничого приміщення для експлуатації ВДТ:

– воно не може бути розміщено у підвалах та цокольних поверхах;

– площа на одне робоче місце в такому приміщенні становить не менше 6,0 м^2, а об’єм – не менше 20,0 м^3;

– присутнє природне та штучне освітлення;

– наявні шафи для зберігання документів, магнітних дисків, полиці, стелажі, тумби, з урахуванням вимог до площі приміщення;

– щоденно проводиться вологе прибирання.

Поруч з приміщенням для роботи з ВДТ знаходяться обладнані:

– побутова кімната для відпочинку під час роботи;

– кімната психологічного розвантаження.

Штучне освітлення в приміщеннях з робочим місцем, обладнаним ВДТ, має здійснюватись системою загального рівномірного освітлення. Як джерело штучного освітлення мають застосовуватись люмінесцентні лампи [22].

Вимоги до освітлення приміщень та робочих місць під час роботи з ВДТ:

– освітленість на робочому місці відповідає характеру зорової роботи;

– забезпечується достатньо рівномірне розподілення яскравості на робочій поверхні монітора, а також в межах навколишнього простору;

– на робочій поверхні відсутні різкі тіні;

– в полі зору відсутні відблиски;

– величина освітленості є постійною під час роботи;

– спрямованість світлового потоку і необхідний склад світла знаходяться на оптимальному рівні.

Застосування світильників без розсіювачів та екрануючих гратів заборонено.

3.4 Висновки до третього розділу

В третьому розділі кваліфікаційної роботи описані питання з безпеки життєдіяльності та охорони праці.

В першому питанні розглядається долікарська допомога при вивихах, що є важливим питанням, оскільки розроблений додаток призначений для інструкторів студії танців «SDC». Танці – це фізична активність, внаслідок якої часто може виникнути перелом. Саме заради додаткового інформування про переломи, їх ознаки та засоби долікарської допомоги було обране це запитання.

В другому питанні описані методи оцінки соціальної та соціально-економічної ефективності заходів щодо покращення умов та охорони праці. Так як мобільний застосунок розробляється для використання інструкторами студії танців «SDC» важливо взяти до уваги методи оцінки ефективності заходів як додатковий соціальний показник, що дозволить надати метрику щодо успішності використання застосунку.

Третє питання покриває інформацію про вимоги до виробничих приміщень для експлуатації ВДТ. Оскільки кваліфікаційна робота бакалавра виконувалась з використанням відео-дисплейного терміналу, були враховані відповідні вимоги щодо робочого місця.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи була проведена розробка мобільного застосунку для студії танців «SDC» на основі клієнт-серверної архітектури під платформу Android, використовуючи мову програмування Kotlin.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

– Проведено аналіз предметної області, тобто досліджено доступні програмні інструменти для розробки мобільного застосунку – мови програмування та інтегровані середовища розробки

– Розглянуто організацію замовника.

– Поставлено вимоги до розробки мобільного застосунку для студії танців «SDC».

– Виділено основні сутності, що присутні в мобільному застосунку для студії танців «SDC».

В другому розділі кваліфікаційної роботи:

– Описано процес проектування, розробки та тестування мобільного застосунку для студії танців «SDC».

– Проведено аналіз варіантів використання та архітектури застосунку.

– Розроблено мобільний застосунок для студії танців «SDC» на основі клієнт-серверної архітектури.

– Описано реалізовані програмні рішення як клієнтської, так і серверної частини.

– Протестовано та звалідовано мобільний застосунок.

У розділі «Безпека життєдіяльності, основи охорони праці» розглянуто питання долікарської допомоги при переломах, методи оцінки соціальної та соціально-економічної ефективності заходів щодо покращення умов та охорони праці і вимоги до виробничих приміщень для експлуатації ВДТ.

ПЕРЕЛІК ДЖЕРЕЛ

1 Android Studio – Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Android_Studio – Дата доступу: 24.02.2021.

2 Android Studio vs IntelliJ IDEA | TrustRadius [Електронний ресурс] – Режим доступу до ресурсу: https://www.trustradius.com/compare-products/android-studio-vs-intellij-idea – Дата доступу: 16.02.2021.

3 API – Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/API – Дата доступу: 19.02.2021.

4 Configure your build | Android Developers [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio/build – Дата доступу: 14.04.2021.

5 Flask [Електронний ресурс] – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/1.1.x/ – Дата доступу: 22.02.2021.

6 HTML5 vs Native Android App – Android Authority [Електронний ресурс] – Режим доступу до ресурсу: https://bit.ly/3xEsyDE – Дата доступу: 22.02.2021.

7 Improve your code with lint checks | Android Developers [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio/write/lint#manuallyRunInspections – Дата доступу: 16.04.2021.

8 IntelliJ IDEA – Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/IntelliJ_IDEA – Дата доступу: 24.02.2021.

9 IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/idea/ – Дата доступу: 24.02.2021.

10 Java – Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Java – Дата доступу: 22.02.2021.

11 Java vs Kotlin [Електронний ресурс] – Режим доступу до ресурсу: https://www.educba.com/java-vs-kotlin/ – Дата доступу: 14.02.2021.

12 Kotlin – Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Kotlin – Дата доступу: 22.02.2021.

13 Kotlin docs [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org/docs/home.html – Дата доступу: 30.03.2021.

14 Kotlin vs Java Comparison [Електронний ресурс] – Режим доступу до ресурсу: https://www.xenonstack.com/blog/kotlin-andriod/ – Дата доступу: 01.04.2021

15 Navigation Architecture | Android Developers [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/guide/navigation – Дата доступу: 26.02.2021.

16 Sdcternopil at Taplink [Електронний ресурс] – Режим доступу до ресурсу: https://taplink.cc/sdcternopil – Дата доступу: 08.04.2021.

17 Unified Modeling Language – Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Unified_Modeling_Language – Дата доступу: 09.04.2021.

18 Діаграма прецедентів – Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Діаграма_прецедентів – Дата доступу: 09.04.2021.

19 Клієнт-серверна архітектура – Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Клієнт-серверна_архітектура – Дата доступу: 04.02.2021.

20 Методика оцінки соціальної і соціально-економічної ефективності заходів щодо покращення охорони праці – Pidru4niki [Електронний ресурс] – Режим доступу до ресурсу: https://bit.ly/2TBA6bb – Дата доступу: 13.05.2021.

21 Організація робочого місця користувача ПК [Електронний ресурс] – Режим доступу до ресурсу: https://bit.ly/3qdy912 – Дата доступу: 06.05.2021.

22 Особливості умов праці з використанням ВДТ [Електронний ресурс] – Режим доступу до ресурсу: https://studopedia.com.ua/1_59409_osoblivosti-umov-pratsi-z-vikoristannyam-peom.html – Дата доступу: 06.05.2021.

23 Переломи кісток – Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Переломи_кісток – Дата доступу: 03.05.2021.

24 Перша допомога при переломах [Електронний ресурс] – Режим доступу до ресурсу: https://medikom.ua/pervaya-pomoshch-pri-perelomah/ – Дата доступу: 03.05.2021.

25 Планування і оснащення свого робочого місця [Електронний ресурс] – Режим доступу до ресурсу: https://studfiles.net/preview/5740410/page:2/ – Дата доступу: 06.05.2021.

26 Різновид мобільних додатків та особливості їх розробки [Електронний ресурс] – Режим доступу до ресурсу: https://gahov.com/university/types-of-apps/ – Дата доступу: 31.03.2021.

27 Розробка мобільних додатків [Електронний ресурс] – Режим доступу до ресурсу: https://ittel.com.ua/informacijni-texnologiyi/rozrobka-mobilnih-dodatkiv/ – Дата доступу: 01.04.2021.

28 Розробка мобільних додатків від А до Я: повний гайд [Електронний ресурс] – Режим доступу до ресурсу: https://dan-it.com.ua/uk/rozrobka-mobilnih-dodatkiv-vid-a-do-ja-povnij-gajd/ – Дата доступу: 06.04.2021.

29 Типи мобільних додатків [Електронний ресурс] – Режим доступу до ресурсу: https://smile-ukraine.com/ua/mobile-apps/mobile-apps-types – Дата доступу: 06.04.2021.

30 Триярусна архітектура – Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Триярусна_архітектура – Дата доступу: 26.02.2021.

ДОДАТКИ

Додаток А

Код клієнтського модуля на серверній частині

Лістинг файлу client.py

```
from functools import wraps

from flask import Blueprint, request, jsonify, current_app
from werkzeug.exceptions import abort
from werkzeug.security import check_password_hash
import uuid
from .db import get_db
import time
import datetime
from .Lessons import Lessons
import logging
from mysql.connector.errors import IntegrityError

bp = Blueprint('client', __name__)
tokens = dict()
lessons = Lessons()
logfile = logging.getLogger('file')


def init_client(func):
    @wraps(func)
    def wrapper(*args, **kws):
        if request.method == "POST":
            token = request.form['token']
            if token in tokens.keys():
                return func(*args, **kws)
            else:
                return "Token is not in the system."
        else:
            return abort(404)

    return wrapper


@bp.route('/client/reg', methods=["POST"])
def register_token():
    if request.method == "POST":
        db = get_db()
        phone = request.form['phone']
        password = request.form['password']
        cursor = db.cursor(dictionary=True, buffered=True)

        error = None
        cursor.execute('SELECT * FROM instructors WHERE
phone=%s', (phone,))
```

```
        user = cursor.fetchone()

        if user is None:
            error = 'Incorrect phone number.'
        elif      not      check_password_hash(user['password'],
password):
            error = 'Incorrect password.'
        elif int(user['valid']) == 0:
            error = 'Non-valid account'

        if error is None:
            token = uuid.uuid1().hex
            tokens[token]    =    {"phone":    phone,    "time":
time.time(), "id": user["id"]}
            cursor.close()
            return    jsonify(message='Success',    token=token,
result=True, error=error)

        cursor.close()
        return       jsonify(message='Failure',       token=None,
result=False, error=error)

    return abort(404)


@bp.route('/client/delete_token', methods=["POST"])
@init_client
def token_delete():
    token = request.form['token']
    tokens.pop(token)
    return "Token deleted"


@bp.route('/client/get_lists', methods=["POST"])
@init_client
def get_lists():
    db = get_db()
    token = request.form['token']
    cursor = db.cursor(dictionary=True, buffered=True)
    cursor.execute('SELECT  *  FROM  instructors  WHERE  id=%s',
(tokens[token]["id"],))
    user = cursor.fetchone()

    cursor.execute('SELECT * FROM instructors where id<>%s and
valid=1', (user['id'],))
    instr = cursor.fetchall()
    instr = [i["instructor_name"] for i in instr]
    instr.sort()
    instr = ["Select", "None"] + instr

    cursor.execute('SELECT  name,  instructors  FROM  sdc_groups
WHERE valid=1')
```

```
    groups = cursor.fetchall()
    groups_name = [i['name'] for i in groups]
    groups_name.sort()
    groups_name.insert(0, "Select")

    specified_groups = list()
    for elem in groups:
        res_instrs = elem["instructors"]
        if not (res_instrs is None or res_instrs == "" or
res_instrs == 'None'):
            res_instrs = list(map(int, res_instrs.split(',')))

            if user['id'] in res_instrs:
                specified_groups.append(elem['name'])

    if specified_groups:
        specified_groups = ['Select'] + specified_groups +
['More']

    cursor.close()
    return jsonify(message="OK", name=user['instructor_name'],
instructors=instr, groups=groups_name,
                   specified_groups=specified_groups)


@bp.route('/client/create', methods=["POST"])
@init_client
def create_lesson():
    def create():
        lesson = dict()
        cursor.execute("SELECT id FROM instructors WHERE
instructor_name=%s", (request.form['instructor1'],))
        instr1_id = cursor.fetchone()["id"]
        if request.form["instructor2"] == "None":
            instr2_id = 0
        else:
            cursor.execute("SELECT id FROM instructors WHERE
instructor_name=%s", (request.form['instructor2'],))
            instr2_id = cursor.fetchone()["id"]
        lesson["time"] = time.strftime("%Y-%m-%d %H:%M")
        lesson["instructors"] = (request.form['instructor1'],
request.form["instructor2"])
        lesson["group"] = request.form['group']
        lesson["clients"] = list()
        lessons[f"{instr1_id}-{instr2_id}"] = lesson

    db = get_db()
    cursor = db.cursor(dictionary=True, buffered=True)

    token = request.form['token']
    instr_id = tokens[token]['id']
    if tokens[token]["id"] not in lessons:
```

```
        create()
        cursor.close()
        return     jsonify(message='Created',     instructors=",
".join(lessons[instr_id]["instructors"]),
group=lessons[instr_id]["group"],
                        time=lessons[instr_id]['time'])
    else:
        submit_lesson(tokens[token]["id"])
        create()
        cursor.close()
        return     jsonify(message='Created',     instructors=",
".join(lessons[instr_id]["instructors"]),
group=lessons[instr_id]["group"],
                        time=lessons[instr_id]['time'])


@bp.route('/client/close_lesson', methods=["POST"])
@init_client
def close_lesson():
    token = request.form['token']
    submit_lesson(tokens[token]["id"])
    return jsonify(message="Lesson closed")


def submit_lesson(instr_id):
    global lessons
    db = get_db()
    cursor = db.cursor(dictionary=True, buffered=True)

    cursor.execute("SELECT    id    FROM    instructors    WHERE
instructor_name=%s", (lessons[instr_id]["instructors"][0],))
    i1_id = cursor.fetchone()['id']
    if lessons[instr_id]["instructors"][1] != "None":
        cursor.execute("SELECT   id   FROM   instructors   WHERE
instructor_name=%s", (lessons[instr_id]["instructors"][1],))
        i2_id = cursor.fetchone()['id']
    else:
        i2_id = 0

    cursor.execute("SELECT * FROM abonement_info")
    abon_info = cursor.fetchall()

    for client in lessons[instr_id]["clients"]:
        cursor.execute("SELECT  id  FROM  users  WHERE  code=%s",
(client,))
        user_id = cursor.fetchone()['id']
        cursor.execute("SELECT  current_number,  abon_number,  id,
abonement_type  FROM  abonement  WHERE  user_id=%s  AND  valid=1",
(user_id,))
        used = cursor.fetchone()
        cursor.execute(
```

```
            "INSERT INTO records(sdc_group, user_id, used,
abon_number, record_date, instructor1_id, instructor2_id,
abon_id                          VALUES(%s,%s,%s,%s,%s,%s,%s,%s)",
(lessons[instr_id]["group"], user_id, used["current_number"],
used['abon_number'],

lessons[instr_id]["time"], i1_id, i2_id, used['id'],))
        used_type = int(used['abonement_type']) - 1

        if     abon_info[used_type]['max_lessons_amount']     ==
used['current_number']:
            cursor.execute("UPDATE abonement SET valid=0 WHERE
user_id=%s        AND        valid=1        AND        id=%s",
(user_id, used['id'],))
        else:
            cursor.execute("UPDATE         abonement         SET
current_number=%s WHERE user_id=%s AND valid=1 AND id=%s",
(used["current_number"] + 1, user_id, used['id'], ))
    db.commit()
    cursor.close()
    del lessons[instr_id]


@bp.route('/client/get_lesson', methods=["POST"])
@init_client
def get_lesson():
    instr_id = tokens[request.form['token']]['id']
    if lessons[instr_id]:
        db = get_db()
        cursor = db.cursor(dictionary=True, buffered=True)

        clients = list()
        for code in lessons[instr_id]['clients']:
            if code == '0000':
                continue
            cursor.execute("SELECT    CONCAT(user_name,    '-',
LPAD(card_number, 4, 0)) AS name FROM users WHERE code=%s",
(code,))
            clients.append(cursor.fetchone()['name'])
        if not lessons[instr_id]['clients']:
            cursor.close()
            return       jsonify(message='Lesson       exists.',
instructors=",          ".join(lessons[instr_id]["instructors"]),
time=lessons[instr_id]['time'],
group=lessons[instr_id]["group"], clients=list(),
                            abons=list())

        users_ids = list()
        for code in lessons[instr_id]['clients']:
            if code == '0000':
                continue
```

```
            cursor.execute("SELECT id FROM users WHERE code=%s",
(code,))
            users_ids.append(cursor.fetchone()['id'])

        abon_info = list()
        temp_users = list()
        for user_id in users_ids:
            if user_id not in temp_users:
                cursor.execute("SELECT                CONCAT('№',
ab.abon_number,'        ',        ab.current_number,        '/',
ab_in.max_lessons_amount)AS info FROM abonement ab LEFT JOIN
abonement_info   ab_in   ON   ab.abonement_type=ab_in.id   WHERE
user_id=%s AND ab.valid=1", (user_id,))
                abon_info.append(cursor.fetchone()['info'])
                temp_users.append(user_id)
                continue

            cursor.execute("SELECT                ab.abon_number,
ab.current_number,  ab_in.max_lessons_amount  FROM  abonement  ab
LEFT  JOIN  abonement_info  ab_in  ON  ab.abonement_type=ab_in.id
WHERE ab.valid=1 AND ab.user_id=%s", (user_id,))
            abonements = cursor.fetchall()
            count = temp_users.count(user_id)

            for abon in abonements:
                if    count    +    abon['current_number']    >
abon['max_lessons_amount']:
                    count    -=    abon['max_lessons_amount']    -
abon['current_number'] + 1
                    continue
                else:
                    info        =        f"№{abon['abon_number']}
{abon['current_number'] + count}/{abon['max_lessons_amount']}"
                    abon_info.append(info)
                    temp_users.append(user_id)
                    break

        if '0000' in lessons[instr_id]['clients']:
            for code in lessons[instr_id]['clients']:
                if code != '0000':
                    continue
                cursor.execute("SELECT   user_name   FROM   users
WHERE code=%s", (code,))
                clients.append(cursor.fetchone()["user_name"])
            cursor.execute("SELECT  abon_number  FROM  abonement
WHERE  user_id=(SELECT  id  FROM  users  WHERE  code='0000')  AND
valid=1")
            single_abon_number                                  =
int(cursor.fetchone()['abon_number'])
            abon_info += [f"№{int(single_abon_number) + n} 1/1"
for n in range(lessons[instr_id]['clients'].count("0000"))]
```

```
        info_z = zip(clients, abon_info)
        info_l = sorted(info_z)
        info = list(zip(*info_l))
        cursor.close()
        return          jsonify(message='Lesson          exists.',
clients=list(info[0]),    abons=list(info[1]),    instructors=",
".join(lessons[instr_id]["instructors"]),
time=lessons[instr_id]['time'],
group=lessons[instr_id]["group"])
    else:
        return jsonify(message="Doesn't exist.")


@bp.route('/client/info', methods=["POST"])
@init_client
def get_info():
    code = request.form["code"]
    db = get_db()
    cursor = db.cursor(dictionary=True, buffered=True)

    cursor.execute("SELECT * FROM users WHERE code=%s", (code,))
    client = cursor.fetchone()
    if client is None:
        cursor.close()
        return   jsonify(message="No   user   with   such   code",
code=code)
    info     =     f"Name:     {client['user_name']}     \nPhone:
{client['phone']} \nCard №{str(client['card_number']).zfill(4)}"
    cursor.execute("SELECT * FROM abonement WHERE user_id=%s AND
valid=1", (client['id'],))
    abonement = cursor.fetchone()

    last_abon = False
    cursor.execute("SELECT * FROM abonement_info WHERE valid=1")
    abonements_list     =     [f"Type     {abon['id']}     with
{abon['max_lessons_amount']} amount of lessons for
{abon['duration']}   days   ({abon['price']}₴)"   for   abon   in
cursor.fetchall()]

    if abonement is None:
        last_abon = True

        cursor.execute("SELECT * FROM abonement WHERE user_id=%s
ORDER BY id DESC", (client['id'],))
        abonement = cursor.fetchone()

        info += "\n\nLast closed abonement info:"

    if abonement is None:
        cursor.close()
        return  jsonify(message="No  active  abonement  for  such
code", code=code, abon_array=abonements_list, info=info)
```

```
    cursor.execute("SELECT max_lessons_amount, duration FROM
abonement_info WHERE id=%s", (abonement['abonement_type'],))
    type_abon = cursor.fetchone()
    info += f"\nType: {abonement['abonement_type']} \nAbonement
number: {abonement['abon_number']} \nCurrent number of training:
{abonement['current_number']-1 if not last_abon else
abonement['current_number']}/{type_abon['max_lessons_amount']} "
\f"\nStarting date: {abonement['starting_date'].strftime('%Y-%m-
%d')} \nEnding date:
{(abonement['starting_date']+datetime.timedelta(days=type_abon['
duration'] + abonement['additional_days'])).strftime('%Y-%m-
%d')}"

    if last_abon:
        cursor.close()
        return jsonify(message="No active abonement for such
code", code=code, abon_array=abonements_list, info=info)

    cursor.close()
    return jsonify(message="Info", info=info, code=code,
abon_array=abonements_list)


def add_existing(code, instr_id):
    db = get_db()
    cursor = db.cursor(dictionary=True, buffered=True)

    cursor.execute("SELECT * FROM users WHERE code=%s", (code,))
    client = cursor.fetchone()

    cursor.execute("SELECT (SUM(ab_i.max_lessons_amount) -
SUM(ab.current_number) + COUNT(*)) as number FROM abonement ab
LEFT JOIN abonement_info ab_i ON ab.abonement_type=ab_i.id WHERE
user_id=%s and ab.valid=1", (client['id'], ))
    possible_amount = cursor.fetchone()['number']

    cursor.execute("SELECT * FROM abonement_info WHERE valid=1")
    abonements_list = [f"Type {abon['id']} with
{abon['max_lessons_amount']} amount of lessons for
{abon['duration']} days ({abon['price']}₴)" for abon in
cursor.fetchall()]

    clients = lessons[instr_id]['clients']
    if clients.count(code) + 1 > possible_amount:
        cursor.close()
        return jsonify(message="No active abonement for such
code", code=code, abon_array=abonements_list)
    else:
        cursor.execute("SELECT ab.id, ab.current_number,
ab.abon_number, ab_i.max_lessons_amount, ab.starting_date,
ab_i.duration, ab.additional_days FROM abonement ab LEFT JOIN
```

```
abonement_info    ab_i    ON    ab.abonement_type=ab_i.id    WHERE
ab.user_id=%s AND ab.valid=1", (client['id'], ))
        abonements = cursor.fetchall()

        now = datetime.date.today()
        days_passed            =            abs((now            -
abonements[0]["starting_date"]).days)
        if              abonements[0]['duration']              +
abonements[0]['additional_days'] < days_passed:
            cursor.execute("UPDATE  abonement  SET  valid=0  WHERE
id=%s", (abonements[0]["id"],))
            db.commit()
            cursor.close()
            return jsonify(message="No active abonement for such
code", code=code, abon_array=abonements_list)

        count = clients.count(code)

        for abon in abonements:
            if     count     +     abon['current_number']     >
abon['max_lessons_amount']:
                #  current_app.logger.info(("Looking  at  another
abonement", count, abon['max_lessons_amount']))
                count     -=     abon['max_lessons_amount']     -
abon['current_number'] + 1
                continue
            else:
                abon_info       =       f"№{abon['abon_number']}
{abon['current_number'] + count}/{abon['max_lessons_amount']}"
                clients = lessons[instr_id]['clients']
                clients.append(code)
                lessons[instr_id]["clients"] = clients
                cursor.close()
                return  jsonify(message="Added",  result="Added",
name=f'{client["user_name"]}-
{str(client["card_number"]).zfill(4)}', abon_info=abon_info)

    current_app.logger.info("Adding existing, not all cases were
checked.")
    cursor.close()
    return jsonify(message='Error. Something went wrong.')


def adder(code, instr_id):
    db = get_db()
    cursor = db.cursor(dictionary=True, buffered=True)

    cursor.execute("SELECT * FROM users WHERE code=%s", (code,))
    client = cursor.fetchone()
    if client is None:
        cursor.close()
```

```
        return jsonify(message="No user with such code",
code=code)

    cursor.execute("SELECT * FROM abonement WHERE user_id=%s AND
valid=1", (client['id'],))
    abonement = cursor.fetchone()
    if abonement is None:
        cursor.execute("SELECT * FROM abonement_info WHERE
valid=1")
        abonements_list = [f"Type {abon['id']} with
{abon['max_lessons_amount']} amount of lessons for
{abon['duration']} days ({abon['price']}₴)" for abon in
cursor.fetchall()]
        cursor.close()
        return jsonify(message="No active abonement for such
code", code=code, abon_array=abonements_list)

    if lessons[instr_id]:
        if code in lessons[instr_id]["clients"]:
            cursor.close()
            return add_existing(code=code, instr_id=instr_id)

        cursor.execute("SELECT max_lessons_amount, duration FROM
abonement_info WHERE id=%s", (abonement['abonement_type'],))
        type_abon = cursor.fetchone()

        now = datetime.date.today()
        days_passed = abs((now -
abonement["starting_date"]).days)

        if type_abon["max_lessons_amount"] >=
abonement["current_number"] and \
                type_abon["duration"] +
abonement['additional_days'] >= days_passed:
            clients = lessons[instr_id]['clients']
            clients.append(code)
            lessons[instr_id]["clients"] = clients
            abon_info = f"№{abonement['abon_number']}
{abonement['current_number']}/{type_abon['max_lessons_amount']}"
            cursor.close()
            return jsonify(message="Added",
name=f'{client["user_name"]}-
{str(client["card_number"]).zfill(4)}', abon_info=abon_info)
        else:
            cursor.execute("UPDATE abonement SET valid=0 WHERE
id=%s", (abonement["id"],))
            db.commit()
            abonements_list = [f"Type {abon['id']} with
{abon['max_lessons_amount']} amount of lessons for
{abon['duration']} days ({abon['price']}₴)" for abon in
cursor.fetchall()]
            cursor.close()
```

```
            return jsonify(message="No active abonement for such
code", code=code, abon_array=abonements_list)
    else:
        current_app.logger.error("TRYING TO WRITE TO EMPTY
LESSON")
        return jsonify(message='ERROR')


@bp.route('/client/add', methods=["POST"])
@init_client
def add_client():
    code = request.form["code"]
    instr_id = tokens[request.form['token']]["id"]
    return adder(code, instr_id)


@bp.route('/client/create_user', methods=["POST"])
@init_client
def create_user():
    db = get_db()
    cursor = db.cursor(dictionary=True)
    try:
        cursor.execute("INSERT INTO users(user_name, phone,
code,  card_number)  VALUES  (%s,  %s,  %s,  %s)",
(request.form['name'],                    request.form["phone"],
request.form['code'], request.form['card_number'], ))
        db.commit()
        cursor.execute("SELECT * FROM abonement_info WHERE
valid=1")
        abonements_list  =  [f"Type  {abon['id']}  with
{abon['max_lessons_amount']}  amount  of  lessons  for
abon['duration']}  days  ({abon['price']}₴)"  for  abon  in
cursor.fetchall()]
    except IntegrityError as e:
        cursor.close()
        return jsonify(message='User exists')
    cursor.close()
    return jsonify(message="Created", code=request.form["code"],
abon_array=abonements_list)


@bp.route('/client/create_abon', methods=["POST"])
@init_client
def create_abon():
    token = request.form['token']
    db = get_db()
    cursor = db.cursor(dictionary=True, buffered=True)

    code = request.form['code']
    instr_id = tokens[token]["id"]

    cursor.execute("SELECT * FROM users WHERE code=%s", (code,))
```

```
    client = cursor.fetchone()
    cursor.execute("SELECT   IFNULL(MAX(abon_number),   0)+1   AS
number FROM abonement WHERE user_id=%s", (client['id'], ))
    next_number = cursor.fetchone()['number']

    cursor.execute("INSERT     INTO     abonement(abonement_type,
starting_date, user_id, instr_id, abon_number) VALUES (%s, %s,
%s,  %s,  %s)",  (request.form['type'],  time.strftime("%Y-%m-%d"),
client['id'], instr_id, next_number, ))
    cursor.execute("SELECT abon_number, id FROM abonement WHERE
user_id=%s AND valid=1", (client['id'],))
    abon = cursor.fetchone()
    cursor.execute("INSERT    INTO    finances(transaction_date,
user_id, instr_id, amount, abon_number, abon_id) VALUES (%s, %s,
%s,  (SELECT  price  FROM  abonement_info  WHERE  id=%s),  %s,  %s)",
(time.strftime("%Y-%m-%d"),        client['id'],        instr_id,
request.form["type"], next_number, abon['id'],))
    db.commit()
    if request.form["mode"] == "True":
        if code in lessons[instr_id]["clients"]:
            cursor.close()
            return add_existing(code=code, instr_id=instr_id)

        clients = lessons[instr_id]['clients']
        clients.append(code)
        lessons[instr_id]["clients"] = clients

        cursor.execute("SELECT CONCAT('№', ab.abon_number, ' ',
ab.current_number,  '/',  SELECT  ab_in.max_lessons_amount  FROM
abonement_info ab_in WHERE ab_in.id=ab.abonement_type)) AS info
FROM    abonement    ab    WHERE    user_id=%s    AND    valid=1",
(client['id'],))
        abon_info = cursor.fetchone()['info']
        cursor.close()
        return     jsonify(message='Created',     result="Added",
name=f'{client["user_name"]}-
{str(client["card_number"]).zfill(4)}',      abon_info=abon_info,
phone=client["phone"])
    cursor.close()
    return jsonify(message="Created", result="Empty")


@bp.route('/client/delete', methods=["POST"])
@init_client
def delete():
    db = get_db()
    instr_id = tokens[request.form['token']]["id"]
    cursor = db.cursor(dictionary=True, buffered=True)

    clients = list()
    for code in lessons[instr_id]['clients']:
```

```
        cursor.execute("SELECT user_name FROM users WHERE
code=%s", (code,))
        clients.append(cursor.fetchone()["user_name"])
    try:
        index = clients.index(request.form['name'].split('-
')[0])
        del (lessons[instr_id]["clients"][index])
        cursor.close()
        return jsonify(message="Deleted")
    except ValueError:
        cursor.close()
        return jsonify(message="No user with such name in
lesson")


@bp.route("/client/get_users", methods=["POST"])
@init_client
def get_users():
    db = get_db()
    cursor = db.cursor(dictionary=True)

    cursor.execute("SELECT user_name FROM users")
    clients = cursor.fetchall()
    all_names = [client["user_name"] for client in clients if
client['user_name'] != 'Unknown']
    all_names.sort()
    cursor.close()
    return jsonify(message="Send", all_names=all_names)


@bp.route("/client/find_code", methods=["POST"])
@init_client
def find_code():
    db = get_db()
    cursor = db.cursor(dictionary=True, buffered=True)

    name = request.form['name']
    cursor.execute("SELECT code FROM users WHERE user_name=%s",
(name,))
    code = cursor.fetchone()["code"]
    instr_id = tokens[request.form['token']]["id"]
    cursor.close()
    return adder(code, instr_id)


@bp.route("/client/get_code", methods=["POST"])
@init_client
def get_code():
    db = get_db()
    cursor = db.cursor(dictionary=True, buffered=True)

    name = request.form['name']
```

```
    cursor.execute("SELECT code FROM users WHERE user_name=%s",
(name,))
    code = cursor.fetchone()["code"]

    cursor.execute("SELECT * FROM abonement_info WHERE valid=1")
    abonements_list     =     [f"Type     {abon['id']}     with
{abon['max_lessons_amount']}     amount     of     lessons     for
{abon['duration']}   days   ({abon['price']}₴)"   for   abon   in
cursor.fetchall()]
    cursor.close()
    return           jsonify(message='Code',           code=code,
abon_array=abonements_list)


@bp.route("/client/get_lessons_dates", methods=["POST"])
@init_client
def get_lessons_dates():
    db = get_db()
    cursor = db.cursor(dictionary=True)

    instr_id = tokens[request.form['token']]["id"]
    cursor.execute("SELECT DISTINCT(SUBSTR(record_date, 1, 10))
AS   record_date   FROM   records   WHERE   (instructor1_id=%s   OR
instructor2_id=%s)", (instr_id, instr_id,))
    dates = cursor.fetchall()
    dates = [date["record_date"] for date in dates]
    cursor.close()
    return jsonify(message="Have dates", dates=dates)


def form_lesson(old_lesson, i_id):
    db = get_db()
    cursor = db.cursor(dictionary=True, buffered=True)

    old_clients = list()
    for row in old_lesson:
        cursor.execute("SELECT user_name FROM users WHERE id =
%s", (row["user_id"],))
        old_clients.append(cursor.fetchone()["user_name"])
    old_abons = list()
    old_date = old_lesson[0]["record_date"]
    whole_sum = 0
    for row in old_lesson:
        cursor.execute("SELECT      max_lessons_amount      FROM
abonement_info  WHERE  id=(SELECT  abonement_type  FROM  abonement
WHERE   user_id=%s   AND   abon_number=%s)",   (row["user_id"],
row['abon_number'],))
        max_lessons_amount                                    =
cursor.fetchone()['max_lessons_amount']
        if str(row["used"]) == "1":
```

```
            cursor.execute("SELECT       amount,       instr_id,
transaction_date    FROM    finances    WHERE    user_id=%s    AND
abon_number=%s", (row["user_id"], row["abon_number"],))
            finance = cursor.fetchone()
            old_info          =          f"№{row['abon_number']}
{row['used']}\\{max_lessons_amount}"

            if    i_id    ==    int(finance["instr_id"])    and
old_date.date() == finance["transaction_date"]:
                old_info += f" \n {finance['amount']}₴"
                whole_sum += float(finance['amount'])
        else:
            old_info          =          f"№{row['abon_number']}
{row['used']}\\{max_lessons_amount}"
        old_abons.append(old_info)
    old_group = old_lesson[0]["sdc_group"]

    cursor.execute("SELECT   instructor_name   FROM   instructors
WHERE   id   IN   (%s,   %s)",   (old_lesson[0]["instructor1_id"],
old_lesson[0]["instructor2_id"],))
    instructors = cursor.fetchall()
    instructors = ", ".join([row["instructor_name"] for row in
instructors])
    whole_sum = f"{whole_sum}₴"
    cursor.close()
    return  jsonify(message="Old  lesson",  clients=old_clients,
abons=old_abons,       whole_sum=whole_sum,       date=old_date,
group=old_group, instructors=instructors)


@bp.route("/client/get_history", methods=["POST"])
@init_client
def get_history():
    db = get_db()
    cursor = db.cursor(dictionary=True, buffered=True)

    date = request.form['date'].split('-')
    instr_id = tokens[request.form['token']]["id"]
    cursor.execute("SELECT  COUNT(DISTINCT(sdc_group))  AS  count
FROM      records      WHERE      YEAR(record_date)=%s      AND
MONTH(record_date)=%s      AND      DAY(record_date)=%s      AND
(instructor1_id=%s  OR  instructor2_id=%s)",  (date[0],  date[1],
date[2], instr_id, instr_id,))
    lessons_amount = cursor.fetchone()["count"]

    if lessons_amount == 1:
        cursor.execute("SELECT    *    FROM    records    WHERE
YEAR(record_date)=%s      AND      MONTH(record_date)=%s      AND
DAY(record_date)=%s       AND       (instructor1_id=%s       OR
instructor2_id=%s)",   (date[0],   date[1],   date[2],   instr_id,
instr_id,))
        old_lesson = cursor.fetchall()
```

```
            cursor.close()
            return form_lesson(old_lesson, instr_id)

        elif lessons_amount > 1:
            if "sdc_group" in request.form.keys():
                cursor.execute("SELECT * FROM records WHERE
YEAR(record_date)=%s AND MONTH(record_date)=%s AND
DAY(record_date)=%s AND (instructor1_id=%s OR instructor2_id=%s)
AND sdc_group LIKE %s", (date[0], date[1], date[2], instr_id,
instr_id, request.form["sdc_group"]))
                old_lesson = cursor.fetchall()
                cursor.close()
                return form_lesson(old_lesson, instr_id)
            else:
                cursor.execute("SELECT DISTINCT(sdc_group) FROM
records WHERE YEAR(record_date)=%s AND MONTH(record_date)=%s AND
DAY(record_date)=%s AND (instructor1_id=%s OR
instructor2_id=%s)", (date[0], date[1], date[2], instr_id,
instr_id,))
                groups_list = cursor.fetchall()
                groups_list = [group["sdc_group"] for group in
groups_list]
                cursor.close()
                return jsonify(message="More than one group",
groups=groups_list)
        return jsonify(message='Error. Nothing returned.')


@bp.route("/client/get_reports", methods=["POST"])
@init_client
def get_reports():
    db = get_db()
    cursor = db.cursor(dictionary=True)
    date = request.form['date'].split('-')
    instr_id = tokens[request.form['token']]["id"]
    whole_sum = 0

    cursor.execute("SELECT transaction_date, SUM(amount) as
money FROM finances WHERE instr_id=%s AND
YEAR(transaction_date)=%s AND MONTH(transaction_date)=%s GROUP
BY transaction_date", (instr_id, date[0], date[1],))
    money_list = cursor.fetchall()
    reports = list()
    for money in money_list:
        reports.append(f"{money['transaction_date']} \t —— \t
{money['money']}₴")
        whole_sum += float(money['money'])
    whole_sum = f"{str(whole_sum)}₴"
    if not reports:
        reports.append("\tNo money for chosen date")
    cursor.close()
```

```
    return      jsonify(message="Reports",      reports=reports,
whole_sum=whole_sum)


@bp.route("/client/get_salaries", methods=["POST"])
@init_client
def get_salaries():
    db = get_db()
    cursor = db.cursor(dictionary=True)

    date = request.form['date'].split('-')
    instr_id = tokens[request.form['token']]["id"]

    cursor.execute("SELECT      sdc_group,      instructor1_id,
instructor2_id,  record_date,  COUNT(user_id)  as  amount  FROM
records WHERE YEAR(record_date)=%s AND MONTH(record_date)=%s AND
(instructor1_id=%s OR instructor2_id=%s) GROUP BY record_date,
sdc_group,  instructor1_id,  instructor2_id ORDER BY sdc_group,
record_date", (date[0], date[1], instr_id, instr_id, ))
    salary_res = cursor.fetchall()
    salary_list    =    [[s["record_date"],    s["sdc_group"],
s["instructor1_id"], s["instructor2_id"], s["amount"]] for s in
salary_res]
    reports = list()
    group = None
    whole_sum = 0
    cursor.execute("SELECT * FROM salary_info")
    salary_info = cursor.fetchall()

    for elem in salary_list:
        price = None
        for i in range(len(salary_info)-1):
            el = salary_info[i]
            next_el = salary_info[i+1]
            sal_date = elem[0].date()

            el_date = el["starting_date"]
            next_el_date = next_el["starting_date"]
            if  price  is  None  and  el_date  <  sal_date  <
next_el_date:
                price = el["price"]
                break
        if price is None:
            sal_date = elem[0].date()
            el_date = salary_info[-1]["starting_date"]
            if el_date <= sal_date:
                price = salary_info[-1]["price"]
        if price is None:
            cursor.close()
            return jsonify(message="Salary error!!")
```

```
        if elem[2] in [0, None, "None"] or elem[3] in [0, None,
"None"]:
            price *= 2
        if group is None or elem[1] != group:
            group = elem[1]
            reports.append(f"\n \t\t {group}:")
        reports.append(" \t — \t ".join([elem[0].strftime("%Y-
%m-%d %H:%M"), str(elem[-1]), f"{price*elem[-1]}₴"]))
        whole_sum += float(price)*elem[-1]
    whole_sum = f"{str(whole_sum)}₴"
    if not reports:
        reports.append("\tNo salary for chosen date")
    cursor.close()
    return      jsonify(message="Reports",      reports=reports,
whole_sum=whole_sum)


@bp.route("/client/get_all_users_info", methods=["POST"])
@init_client
def get_all_users_info():
    db = get_db()
    cursor = db.cursor(dictionary=True)

    cursor.execute("SELECT      users.id,      users.user_name,
users.phone,   ab.valid,   ab.starting_date,   ab.abonement_type,
ab.abon_number, ab.valid, ab.current_number, ab.additional_days,
users.card_number  FROM  users  LEFT  JOIN  abonement  ab  ON
users.id=ab.user_id AND ab.valid <> 0 WHERE code <> '0000' ORDER
BY users.user_name")
    users = cursor.fetchall()
    user_list = list()
    for row in users:
        r    =    f"{row['user_name']}    -    {row['phone']}    -
№{str(row['card_number']).zfill(4)}"
        if row["starting_date"] and row['starting_date'] is not
None:
            abon_date = row["starting_date"]
            cursor.execute("SELECT  max_lessons_amount,  duration
FROM abonement_info WHERE id=%s", (row['abonement_type'],))
            type_abon = cursor.fetchone()
            end_date          =          (abon_date          +
datetime.timedelta(days=type_abon["duration"]                  +
row['additional_days'])).strftime("%Y-%m-%d")
            r      +=      f"      -      №{row['abon_number']}
{row['current_number']-1}/{type_abon['max_lessons_amount']}    -
to: {end_date}"
        user_list.append(r)
    cursor.close()
    return jsonify(message='All users info', users=user_list)


@bp.route("/client/change_user_code", methods=["POST"])
```

```
@init_client
def change_user_code():
    db = get_db()
    cursor = db.cursor(dictionary=True, buffered=True)

    code = request.form["code"]
    row = request.form["row"]

    user_name = row.split(" - ")[0]
    cursor.execute("SELECT id FROM users WHERE user_name=%s",
(user_name,))
    user_id = cursor.fetchone()['id']
    try:
        cursor.execute("UPDATE users SET code=%s WHERE id=%s",
(code, user_id,))
        db.commit()
        cursor.close()
        logfile.info(f"Instructor
{tokens[request.form['token']]['id']} CHANGED USER CODE FOR
user_id {user_id} TO {code}")
        return jsonify(message='Code changed')
    except Exception as e:
        current_app.logger.error(e)
        cursor.close()
        return jsonify(message="Problems occurred")


@bp.route("/client/get_session_hist", methods=['POST'])
def get_session_hist():
    db = get_db()
    cursor = db.cursor(dictionary=True)

    code = request.form["code"]

    cursor.execute("SELECT id FROM abonement WHERE valid=1 AND
user_id=(SELECT id FROM users WHERE code=%s)", (code,))
    abon_id = cursor.fetchall()

    if not abon_id:
        cursor.execute("SELECT id FROM abonement WHERE
user_id=(SELECT id FROM users WHERE code=%s)", (code,))
        abon_id = reversed(cursor.fetchall())
    abon_id = abon_id[0]['id']
    cursor.execute("SELECT record_date, sdc_group,
i.instructor_name, used FROM records LEFT JOIN instructors i ON
instructor1_id=i.id WHERE abon_id=%s", (abon_id, ))
    hist = cursor.fetchall()
    result = ""
    for row in hist:
        result += f"{row['record_date']} - {row['sdc_group']} -
{row['instructor_name']} - Заняття {row['used']}\n\n"
    cursor.close()
```

```
    return jsonify(message="History", history=result[:-2])


@bp.route("/client/add_simple_abon", methods=['POST'])
@init_client
def add_simple_abon():
    db = get_db()
    cursor = db.cursor(dictionary=True, buffered=True)

    code = "0000"
    instr_id = tokens[request.form['token']]["id"]

    cursor.execute("SELECT * FROM users WHERE code=%s", (code,))
    client = cursor.fetchone()
    cursor.execute("SELECT   IFNULL(MAX(abon_number),   0)+1   AS
number FROM abonement WHERE user_id=%s", (client['id'], ))
    next_number = cursor.fetchone()['number']

    cursor.execute("INSERT     INTO     abonement(abonement_type,
starting_date, user_id, instr_id, abon_number) VALUES (%s, %s,
%s,   %s,   %s)",   (1,   time.strftime("%Y-%m-%d"),   client['id'],
instr_id, next_number, ))
    cursor.execute("SELECT abon_number, id FROM abonement WHERE
user_id=%s AND valid=1 ORDER BY id DESC", (client['id'],))
    abon = cursor.fetchone()
    cursor.execute("INSERT    INTO    finances(transaction_date,
user_id, instr_id, amount, abon_number, abon_id) VALUES (%s, %s,
%s, (SELECT price FROM abonement_info WHERE id=%s), %s, %s)",
(time.strftime("%Y-%m-%d"),    client['id'],    instr_id,    1,
abon["abon_number"], abon['id'],))
    db.commit()

    clients = lessons[instr_id]['clients']
    clients.append(code)
    lessons[instr_id]["clients"] = clients
    cursor.execute("SELECT  CONCAT('№',  ab.abon_number,  '  ',
ab.current_number,  '/',  (SELECT  ab_in.max_lessons_amount  FROM
abonement_info ab_in WHERE ab_in.id=ab.abonement_type)) AS info
FROM abonement ab WHERE user_id=%s and valid=1 ORDER BY id
DESC", (client['id'],))
    abon_info = cursor.fetchone()['info']
    cursor.close()
    return   jsonify(message='Added',   name=client["user_name"],
abon_info=abon_info)


@bp.route("/client/update_user", methods=['POST'])
@init_client
def update_user():
    db = get_db()
    cursor = db.cursor(dictionary=True, buffered=True)
```

```
    try:
        cursor.execute("UPDATE users SET user_name=%s, phone=%s,
card_number=%s     WHERE     code=%s",     request.form['name'],
request.form['phone'],              request.form['card_number'],
request.form['code'], ))
        cursor.close()
        logfile.info(f"Instructor
{tokens[request.form['token']]['id']} CHANGE USER INFO WITH CODE
{request.form['code']}    TO    user_name={request.form['name']},
phone={request.form['phone']},
card_number={request.form['card_number']}")
        return jsonify(message='Updated')
    except Exception as e:
        current_app.logger.error(e)
        cursor.close()
        return jsonify(message='Error occurred', error=e)


@bp.route("/client/get_name", methods=['POST'])
@init_client
def get_name():
    db = get_db()
    cursor = db.cursor(dictionary=True, buffered=True)
    code = request.form["code"]

    cursor.execute("SELECT         CONCAT(user_name,         '-',
LPAD(card_number,  4,  0))  AS  name  FROM  users  WHERE  code=%s",
(code,))
    name = cursor.fetchone()
    if name is not None:
        name = name['name']
        return   jsonify(message='Returning   name',   code=code,
name=name)
    else:
        return   jsonify(message='Returning   name',   code=code,
name="")


@bp.route('/abonement/information', methods=["GET", "POST"])
def get_client_info():
    db = get_db()
    code = request.form["code"]
    cursor = db.cursor(dictionary=True, buffered=True)

    cursor.execute("SELECT * FROM users WHERE code=%s", (code,))
    client = cursor.fetchone()
    if client is None:
        cursor.close()
        return   jsonify(message="No   user   with   such   code",
code=code)
    info     =     f"Name:     {client['user_name']}     \nPhone:
{client['phone']} \nCard №{str(client['card_number']).zfill(4)}"
```

```
    cursor.execute("SELECT * FROM abonement WHERE user_id=%s AND
valid=1", (client['id'],))
    abonement = cursor.fetchone()
    if abonement is None:
        cursor.close()
        return jsonify(message="No active abonement for such
code", code=code, info=info)
    cursor.execute("SELECT max_lessons_amount, duration FROM
abonement_info WHERE id=%s",
                           (abonement['abonement_type'],))
    type_abon = cursor.fetchone()
    info += f"\nType: {abonement['abonement_type']} \nAbonement
number: {abonement['abon_number']} " \
            f"\nCurrent number of training:
{abonement['current_number']-
1}/{type_abon['max_lessons_amount']} " \
            f"\nStarting date:
{abonement['starting_date'].strftime('%Y-%m-%d')}" \
            f"\nEnding date:
{(abonement['starting_date']+datetime.timedelta(days=type_abon['
duration'] + abonement['additional_days'])).strftime('%Y-%m-
%d')}"

    cursor.close()
    return jsonify(message="Info", info=info)
```

Додаток Б

Код клієнтського модуля клієнтської частини

Лістинг файлу AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.ebert_l.sdc_app">
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.CAMERA" />
    <application
            android:allowBackup="true"
            android:icon="@mipmap/ic_launcher"
            android:label="@string/app_name"
            android:roundIcon="@mipmap/ic_launcher_round"

android:networkSecurityConfig="@xml/network_security_config"
            android:supportsRtl="true"
            android:theme="@style/AppTheme"
            android:fullBackupContent="@xml/backup_descriptor">
        <activity                       android:name=".MainActivity"
android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category
android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity             android:name=".CustomScannerActivity"
android:screenOrientation="portrait">
        </activity>
    </application>
</manifest>
```

Лістинг файлу MainActivity.kt

```
package com.ebert_l.sdc_app
import android.os.Bundle
import android.view.View
import androidx.appcompat.app.AppCompatActivity
import androidx.navigation.findNavController
import androidx.navigation.fragment.NavHostFragment
import androidx.navigation.ui.setupWithNavController
```

```
import
com.google.android.material.bottomnavigation.BottomNavigationView
import kotlinx.android.synthetic.main.activity_main.*
import okhttp3.OkHttpClient
import java.security.KeyStore
import java.security.cert.Certificate
import java.security.cert.CertificateFactory
import javax.net.ssl.SSLContext
import javax.net.ssl.TrustManagerFactory
import javax.net.ssl.X509TrustManager

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setupNav()
    }

    override fun onBackPressed() {
        val currentDestination =
NavHostFragment.findNavController(nav_host_fragment).currentDestina
tion
        when (currentDestination?.id) {
            R.id.nav_training -> finish()
            R.id.nav_code -> finish()
        }
        super.onBackPressed()
    }

    private fun setupNav() {
        val                    navController                    =
findNavController(R.id.nav_host_fragment)
findViewById<BottomNavigationView>(R.id.bottomNavigationView)
            .setupWithNavController(navController)
        navController.addOnDestinationChangedListener     {     _,
destination, _ ->
            when (destination.id) {
                R.id.loginActivity                                ->
bottomNavigationView.visibility = View.GONE
                else    ->    bottomNavigationView.visibility    =
View.VISIBLE
            }
        }
    }

    fun getSelfCertClient(): OkHttpClient {
        val cf = CertificateFactory.getInstance("X.509")
```

```
        val                          cert                          =
application.resources.openRawResource(R.raw.cert)
        val ca: Certificate = cert.use { cert ->
            cf.generateCertificate(cert)
        }
        val keyStoreType = KeyStore.getDefaultType()
        val keyStore = KeyStore.getInstance(keyStoreType)
        keyStore.load(null, null)
        keyStore.setCertificateEntry("ca", ca)
        val                      tmfAlgorithm                      =
TrustManagerFactory.getDefaultAlgorithm()
        val tmf =
            TrustManagerFactory.getInstance(tmfAlgorithm)
        tmf.init(keyStore)
        val sslContext = SSLContext.getInstance("TLS")
        sslContext.init(null, tmf.trustManagers, null)
        return OkHttpClient.Builder().sslSocketFactory(
            sslContext.socketFactory,
            tmf.trustManagers[0] as X509TrustManager
        ).hostnameVerifier { hostname, session -> hostname in url
}.build()
    }


    companion object {
        var token: String = ""
        const val url: String = "URL"
    }
}
```

Лістинг файлу ClientHTTP.kt

```
from functools import wraps
package com.ebert_l.sdc_app
import android.app.Activity
import android.content.Context
import android.widget.Toast
import okhttp3.*
import org.json.JSONObject
import java.io.IOException
import java.util.*
import java.util.concurrent.CountDownLatch
class ClientHTTP(val activity: Activity?, val context: Context?) {
    var      client:      OkHttpClient      =      (activity      as
MainActivity).getSelfCertClient()
    val countDownLatch = CountDownLatch(1)
```

```
    fun createCallback(func: (jsonObject: JSONObject) -> Unit):
Callback {
        return object : Callback {
            override fun onFailure(call: Call, e: IOException) {
                activity?.runOnUiThread {
                    Toast.makeText(
                        context,
                        "Connecting to the system failed.",
                        Toast.LENGTH_SHORT
                    ).show()
                    Toast.makeText(context,
e.toString(),Toast.LENGTH_SHORT).show()
                }
                countDownLatch.countDown()
            }

      override fun onResponse(call: Call, response: Response) {
                val body = response.body()?.string()
                try {
                    when {
                        body == "Token is not in the system." -> {
                            activity?.runOnUiThread {
                                Toast.makeText(
                                    context,
                                    "Token is not in the system.",
                                    Toast.LENGTH_LONG
                                ).show()
                            }
                        }
body!!.toLowerCase(Locale.ROOT).contains("internal server error") -
> {
                            activity?.runOnUiThread {
                                Toast.makeText(
                                    context,
                                    "Internal Server Error",
                                    Toast.LENGTH_LONG
                                ).show()
                            }
                        }
                        else -> {
                            if
(body.toLowerCase(Locale.ROOT).contains("error")){
                            activity?.runOnUiThread {
                                Toast.makeText(
                                    context,
                                    "Error might be present",
                                    Toast.LENGTH_SHORT
                                ).show()
```

```
                                }
                            }
                            val jsonObject = JSONObject(body)
                            func(jsonObject)
                        }
                    }
                } catch (e: Exception) {
                    println(e)
                    activity?.runOnUiThread {
                        Toast.makeText(
                            context,
                            "Oops. Some problem detected.",
                            Toast.LENGTH_LONG
                        ).show()
                    }
                }
                countDownLatch.countDown()
            }
        }
    }

    fun sendRequest(url: String, parameters: HashMap<String,
String>, callback: Callback): Call {
        val builder = FormBody.Builder()
        val it = parameters.entries.iterator()
        while (it.hasNext()) {
            val pair = it.next() as Map.Entry<*, *>
            builder.add(pair.key.toString(), pair.value.toString())
        }
        val formBody = builder.build()
        val request = Request.Builder().url(url).post(formBody)
            .build()
        val call = client.newCall(request)
        call.enqueue(callback)
        countDownLatch.await()
        return call
    }
}
```

Лістинг файлу TrainFragment1.kt

```
package com.ebert_l.sdc_app
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import androidx.core.os.bundleOf
```

```
import androidx.fragment.app.Fragment
import androidx.navigation.Navigation
import kotlinx.android.synthetic.main.train_fragment1.*
import kotlinx.serialization.json.Json
import kotlinx.serialization.list
import kotlinx.serialization.serializer

class TrainFragment1 : Fragment() {
    var clients: List<String> = listOf("")
    var abons = listOf("")
    var time: String = ""
    var instructors: String = ""
    var group: String = ""
    var navigate = false
    val url = MainActivity.url
    val token = MainActivity.token

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val   view   =   inflater.inflate(R.layout.train_fragment1,
container, false)
        val clientHTTP = ClientHTTP(activity, context)
        clientHTTP.sendRequest("$url/client/get_lesson",
hashMapOf("token" to token), clientHTTP.createCallback {
            navigate = true
            if (it.getString("message") == "Lesson exists."){
                clients    =    Json.parse(String.serializer().list,
it.getString("clients"))
                abons     =     Json.parse(String.serializer().list,
it.getString("abons"))
                time = it.getString("time")
                instructors = it.getString("instructors")
                group = it.getString("group")
activity?.runOnUiThread{view.findViewById<Button>(R.id.loadButton).
isEnabled = true
                }
            }
            })
        return view
    }

    override  fun  onViewCreated(view:  View,  savedInstanceState:
Bundle?) {
        createButton.setOnClickListener {
```

```
Navigation.findNavController(view).navigate(R.id.action_nav_trainin
g_to_trainFragment2) }
        loadButton.setOnClickListener {
            if (navigate){
                clients = clients.toMutableList()
                val     bundle     =     bundleOf("clients"     to
ArrayList(clients), "instructors" to instructors,
                    "time" to time, "group" to group, "abons" to
ArrayList(abons))
Navigation.findNavController(view).navigate(R.id.action_nav_trainin
g_to_trainFragment3, bundle)
            }
        }
        historyButton.setOnClickListener {
Navigation.findNavController(view).navigate(R.id.action_nav_trainin
g_to_historyFragment)
        }
        reportsButton.setOnClickListener {
Navigation.findNavController(view).navigate(R.id.action_nav_trainin
g_to_reportFragment,
                bundleOf("salary" to false))
        }
        salaryButton.setOnClickListener {
Navigation.findNavController(view).navigate(R.id.action_nav_trainin
g_to_reportFragment,
                bundleOf("salary" to true))
        }
        userButton.setOnClickListener {
Navigation.findNavController(view).navigate(R.id.action_nav_trainin
g_to_userFragment)
        }
        super.onViewCreated(view, savedInstanceState)
    }
}
```

Лістинг файлу TrainFragment2.kt

```
package com.ebert_l.sdc_app
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.AdapterView
import android.widget.ArrayAdapter
import androidx.core.os.bundleOf
import androidx.fragment.app.Fragment
import androidx.navigation.Navigation
```

```
import kotlinx.android.synthetic.main.train_fragment2.*
import kotlinx.serialization.json.Json
import kotlinx.serialization.list
import kotlinx.serialization.serializer


class TrainFragment2 : Fragment() {
    var groupsList: List<String> = listOf("Empty")
    var name: String = ""
    var instructorsList: List<String> = listOf("Empty")
    var specifiedGrupsList: List<String> = listOf("Empty")
    var bundle: Bundle = bundleOf()

    val url = MainActivity.url
    val token = MainActivity.token

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val clientHTTP = ClientHTTP(activity, context)
        clientHTTP.sendRequest(
            "$url/client/get_lists",
            hashMapOf("token" to token),
            clientHTTP.createCallback {
                name = it.get("name").toString()
                instructorsList =
                    Json.parse(String.serializer().list,
it.get("instructors").toString())
                groupsList  =  Json.parse(String.serializer().list,
it.get("groups").toString())
                if (it.getString("specified_groups").isNotBlank()){
                    specifiedGroupsList                                =
Json.parse(String.serializer().list,
it.getString("specified_groups"))
                }

    override  fun  onViewCreated(view:  View,  savedInstanceState:
Bundle?) {
        textView.text = getString(R.string.you_are, name)
        val        adapter        =        ArrayAdapter(context!!,
android.R.layout.simple_spinner_item, instructorsList)
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dro
pdown_item)
        spinner.adapter = adapter
        spinner.onItemSelectedListener       =       object       :
AdapterView.OnItemSelectedListener {
            override fun onItemSelected( parent: AdapterView<*>,
view_: View, position: Int, id: Long ) {
```

```
                if (position != 0) {
                    groupSpinnerLayout.visibility = View.VISIBLE
                } else {
                    groupSpinnerLayout.visibility = View.INVISIBLE
                }
            }

          override fun onNothingSelected(parent: AdapterView<*>) {
            }
        }
        val adapter2 = if (specifiedGroupsList.isNotEmpty()){
            ArrayAdapter(context!!,
android.R.layout.simple_spinner_item, specifiedGroupsList)
        } else{
            ArrayAdapter(context!!,
android.R.layout.simple_spinner_item, groupsList)
        }

adapter2.setDropDownViewResource(android.R.layout.simple_spinner_dr
opdown_item)
        spinner2.adapter = adapter2
        spinner2.onItemSelectedListener       =       object       :
AdapterView.OnItemSelectedListener {
      override   fun   onItemSelected(parent:   AdapterView<*>,view_:
View, position: Int, id: Long ) {
                if   (position   ==   adapter2.count   -   1   &&
adapter2.getItem(position) == "More"){
                    adapter2.clear()
                    specifiedGroupsList = groupsList
                    adapter2.addAll(groupsList)
                    adapter2.notifyDataSetChanged()
                    spinner2.setSelection(0)
                }
                if (position != 0 && adapter2.getItem(position) !=
"More") {
                    continueButton.visibility = View.VISIBLE
                    continueButton.setOnClickListener {
                        getMore(group                                =
parent.getItemAtPosition(position).toString(),
                                instructor2                          =
spinner.getItemAtPosition(spinner.firstVisiblePosition).toString())
                    }
                } else {
                    continueButton.visibility = View.INVISIBLE
                }
            }

         override fun onNothingSelected(parent: AdapterView<*>) {
```

```
            }
        }
        super.onViewCreated(view, savedInstanceState)
    }

    fun getMore(group: String, instructor2: String){
        var navigate = false
        val clientHTTP = ClientHTTP(activity, context)
        clientHTTP.sendRequest("$url/client/create", hashMapOf(
            "group" to group,
            "instructor2" to instructor2,
            "instructor1" to name, "token" to token
        ), clientHTTP.createCallback {
            if (it.getString("message") == "Created") {
                bundle = bundleOf(
                    "time" to it.getString("time"),
                    "group" to it.getString("group"),
                    "instructors" to it.getString("instructors")
                )
                navigate = true
            }
        })
        if (navigate) {
            view?.let {
                Navigation.findNavController(it)

.navigate(R.id.action_trainFragment2_to_trainFragment3, bundle)
            }
        }
    }
}
```

Лістинг файлу TrainFragment3.kt

```
package com.ebert_l.sdc_app
import android.content.Intent
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.os.Message
import android.text.Editable
import android.text.TextWatcher
import android.view.*
import android.widget.ArrayAdapter
import androidx.fragment.app.Fragment
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
```

```
import androidx.navigation.Navigation
import androidx.recyclerview.widget.ItemTouchHelper
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.google.zxing.client.android.Intents
import com.google.zxing.integration.android.IntentIntegrator
import kotlinx.android.synthetic.main.choose_user.view.*
import kotlinx.android.synthetic.main.create_user_dialog.*
import kotlinx.android.synthetic.main.create_user_dialog.view.*
import kotlinx.android.synthetic.main.train_fragment3.*
import kotlinx.serialization.json.Json
import kotlinx.serialization.list
import kotlinx.serialization.serializer

class TrainFragment3 : Fragment() {
    private var clients: MutableList<String> = mutableListOf()
    private var abons = mutableListOf<String>()
    val url = MainActivity.url
    val token = MainActivity.token

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle? ): View? {
        val   view   =   inflater.inflate(R.layout.train_fragment3,
container, false)
        val getClients = arguments?.getStringArrayList("clients")
        if (!getClients.isNullOrEmpty()) {
            clients = getClients.toMutableList()
view?.findViewById<RecyclerView>(R.id.recyclerView)?.adapter?.notif
yDataSetChanged()}
        val getAbons = arguments?.getStringArrayList("abons")
        if (!getAbons.isNullOrEmpty()){
            abons = getAbons.toMutableList()
view?.findViewById<RecyclerView>(R.id.recyclerView)?.adapter?.notif
yDataSetChanged() }
        setHasOptionsMenu(true)
        return view
    }

    override   fun   onCreateOptionsMenu(menu:   Menu,   inflater:
MenuInflater) {
        inflater.inflate(R.menu.train_menu, menu)
        super.onCreateOptionsMenu(menu, inflater)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when (item.itemId) {
            R.id.closeOption -> {
```

```
                val clientHttpCl = ClientHTTP(activity, context)
                clientHttpCl.sendRequest(
                    "$url/client/close_lesson",
                    hashMapOf("token" to token),
                    clientHttpCl.createCallback {
                        if   (it.getString("message")   ==   "Lesson
closed") {
                            activity?.runOnUiThread {
                                Toast.makeText(context,     "Lesson
closed", Toast.LENGTH_LONG).show() }
                        }
                    })
                Navigation.findNavController(view!!).popBackStack()
            }

            R.id.chooseOption -> {
                val clientHttpCh = ClientHTTP(activity, context)
                clientHttpCh.sendRequest(
                    "$url/client/get_users",
                    hashMapOf("token" to token),
                    clientHttpCh.createCallback {
                        if (it.getString("message") == "Send") {
                            val arr =
Json.parse(String.serializer().list, it.getString("all_names"))
                            activity?.runOnUiThread                {
createAllUsersDialog(arr) }
                        }
                    })
            }
            R.id.singleOption -> {
                val    clientHTTPSingle    =    ClientHTTP(activity,
context)
clientHTTPSingle.sendRequest("$url/client/add_simple_abon",
                    hashMapOf("token"           to           token),
clientHTTPSingle.createCallback {
                        if(it.getString("message") == "Added") {
                            val newClient = it.getString("name")
                            val             newAbonInfo             =
it.getString("abon_info")
                            clients.add(newClient)
                            abons.add(newAbonInfo)
                            activity?.runOnUiThread {
recyclerView.adapter?.notifyItemInserted(clients.size)
                            }
                        }
                    })
            }
```

```
            else -> return super.onOptionsItemSelected(item)
        }
        return true

    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        timeView.text          =          getString(R.string.date,
arguments?.getString("time"))
        instrView.text      =      getString(R.string.instructors,
arguments?.getString("instructors"))
        groupView.text         =         getString(R.string.group,
arguments?.getString("group"))
        recyclerView.layoutManager = LinearLayoutManager(context)
        recyclerView.adapter = Adapter(clients, abons)
        recyclerView.setHasFixedSize(true)
        val        itemTouchHelper        =        object        :
SwipeToDeleteCallback(context!!) {
            override           fun           onSwiped(viewHolder:
RecyclerView.ViewHolder, direction: Int) {
                createSwipeDialog(viewHolder)
            }
        }

ItemTouchHelper(itemTouchHelper).attachToRecyclerView(recyclerView)
        scanButton.setOnClickListener {
            val                qrScannerIntegrator                =
IntentIntegrator.forSupportFragment(this)
            qrScannerIntegrator.captureActivity                    =
CustomScannerActivity::class.java
            qrScannerIntegrator.addExtra(Intents.Scan.SCAN_TYPE,
Intents.Scan.MIXED_SCAN)
                .setDesiredBarcodeFormats(IntentIntegrator.QR_CODE)
                .setOrientationLocked(false)
                .setBeepEnabled(false)
                .setPrompt("Put a code inside rectangle to scan
it")
            qrScannerIntegrator.initiateScan()
        }
        super.onViewCreated(view, savedInstanceState)
    }

    override fun onActivityResult(requestCode: Int, resultCode:
Int, data: Intent?) {
        try {
            if (data != null) {
```

```
                val                    result                    =
IntentIntegrator.parseActivityResult(requestCode, resultCode, data)
                if (result != null) {
                    val code = result.contents.toString()
                    var presence = abonPresence(code)
                    if (!MainActivity.validateCode(code)) {
                        activity?.runOnUiThread {
                            Toast.makeText(
                                context,
                                "Code is not valid",
                                Toast.LENGTH_LONG
                            ).show()
                        }
                        presence = false
                    }
                    if (presence) {
                        val   clientHTTP   =   ClientHTTP(activity,
context)
                        clientHTTP.sendRequest(
                            "$url/client/add",
                            hashMapOf("token" to token, "code" to
code), clientHTTP.createCallback {
                                when (it.getString("message")) {
                                    "Added" -> {
                     val newClient = it.getString("name")
                     val newAbonInfo = it.getString("abon_info")
                     clients.add(newClient)
                     abons.add(newAbonInfo)
                     activity?.runOnUiThread {
recyclerView.adapter?.notifyItemInserted(clients.size) }
                                    }
              "Already here" -> activity?.runOnUiThread {
                                        Toast.makeText(context,
"Already here", Toast.LENGTH_LONG).show()    }
              "No user with such code" -> activity?.runOnUiThread {
createUserDialog(it.getString("code"))      }
              "No   active   abonement   for   such   code"   ->
activity?.runOnUiThread {
          val    arr    =    Json.parse(String.serializer().list,
it.getString("abon_array"))
createAbonDialog(it.getString("code"), arr)}
                                    }
                                })
                    }
                } else {
     super.onActivityResult(requestCode, resultCode, data)
                }
            }
```

```
        } catch (e: Exception){
            activity!!.runOnUiThread {
                val clientHTTP = ClientHTTP(activity, context)
                clientHTTP.sendRequest("$url/client/log",
hashMapOf("Message" to e.toString()),
                    clientHTTP.createCallback { println(it) })
            Toast.makeText(context,                          "Error",
Toast.LENGTH_SHORT).show()
Navigation.findNavController(view!!).navigate(R.id.action_loginActi
vity2_to_nav_training)
            }
        }
    }

    class Adapter(private val clients: MutableList<String>, private
val abons: MutableList<String>) :
        RecyclerView.Adapter<Adapter.ViewHolder>() {
        override fun getItemCount() = clients.size
        override    fun    onCreateViewHolder(parent:    ViewGroup,
viewType: Int): ViewHolder {
            val itemView =
LayoutInflater.from(parent.context).inflate(R.layout.element_view,
parent, false)
            return ViewHolder(itemView)
        }

        override fun onBindViewHolder(holder: ViewHolder, position:
Int) {
            holder.textView.text = clients[position]
            "${position+1}-".also { holder.indexView.text = it }
            holder.abonView.text = abons[position]
        }

        fun removeAt(position: Int) {
            clients.removeAt(position)
            abons.removeAt(position)
            notifyItemRemoved(position)
            notifyItemRangeChanged(position, clients.size)
        }

        class         ViewHolder(itemView:         View)         :
RecyclerView.ViewHolder(itemView) {
            var           textView:           TextView           =
itemView.findViewById(R.id.list_itemText)
            var           indexView:           TextView           =
itemView.findViewById(R.id.indexView)
            var           abonView:           TextView           =
itemView.findViewById(R.id.abonView)
```

```
        }
    }

    fun createUserDialog(code: String) {
        val                         dialogView                         =
requireActivity().layoutInflater.inflate(R.layout.create_user_dialo
g, null)
        val dialog = AlertDialog.Builder(activity!!)
            .setView(dialogView)
            .setPositiveButton("Create") { _, _ -> }
            .setNegativeButton("Cancel") { dialog, _ ->
                dialog.dismiss()
            }.setTitle("Create user").create()
        dialog.show()

dialog.getButton(AlertDialog.BUTTON_POSITIVE).setOnClickListener  {
itView ->
            val name = dialogView.createNameEdit.text.toString()
            val phone = dialogView.createPhoneEdit.text.toString()
            val                       cardNumber                       =
dialog.createCardNumberEdit.text.toString()
            if   (name.isNotBlank()   &&   phone.isNotBlank()   &&
cardNumber.isNotBlank()) {
                val clientHTTP = ClientHTTP(activity, context)
                clientHTTP.sendRequest("$url/client/create_user",
                    hashMapOf(
                        "token" to token,
                        "code" to code,
                        "name" to name,
                        "phone" to phone,
                        "card_number" to cardNumber
                    ), clientHTTP.createCallback {
                        if (it.getString("message") == "Created") {
                            activity?.runOnUiThread {
                                Toast.makeText(
                                    context,
                                    "User created",
                                    Toast.LENGTH_LONG
                                ).show()
                                val arr =
Json.parse(String.serializer().list, it.getString("abon_array"))
                                createAbonDialog(code, arr)
                            }
                        }
                    })
                dialog.dismiss()
            } else {
                activity?.runOnUiThread {
```

```
                    Toast.makeText(context,  "One  of  fields  is
empty", Toast.LENGTH_LONG).show()
                }
            }
        }
    }

    fun createAbonDialog(code: String, arr: List<String>) {
        AlertDialog.Builder(activity!!)
            .setTitle("Select abonement type")
            .setItems(arr.toTypedArray()) { _, which ->
                val clientHTTP = ClientHTTP(activity, context)
                if (abonAlert(arr[which])){
                    val  hashMap  =  hashMapOf("token"  to  token,
"code"  to  code,  "type"  to  arr[which].split("  ")[1],  "mode"  to
"True")
  clientHTTP.sendRequest("$url/client/create_abon", hashMap,
                        clientHTTP.createCallback {
                            if (it.getString("result") == "Added"){
          val newClient = it.getString("name")
          val newAbonInfo = it.getString("abon_info")
          clients.add(newClient)
          abons.add(newAbonInfo)
          activity?.runOnUiThread {
              Toast.makeText(context,                    "Abonement
created",Toast.LENGTH_LONG ).show()
      recyclerView.adapter?.notifyItemInserted(clients.size)
      Toast.makeText(context,             "User             added",
Toast.LENGTH_LONG).show()}
             }
             })
                } else{
                    activity?.runOnUiThread {
                Toast.makeText(context,  "Picking dismissed",
                        Toast.LENGTH_LONG).show()
                    }
                }
            }
            .setNegativeButton("Cancel"){dialogInterface,    _    ->
dialogInterface.dismiss()}
            .create().show()

    }

    fun createSwipeDialog(viewHolder: RecyclerView.ViewHolder) {
        AlertDialog.Builder(activity!!).setTitle("Delete?")
            .setMessage("\nDo    you    want    to    delete    user
${clients[viewHolder.adapterPosition]}?\n")
```

```
            .setPositiveButton("Yes") { _, _ ->
                val clientHTTP = ClientHTTP(activity, context)
                clientHTTP.sendRequest(
                    "$url/client/delete",
                    hashMapOf("token"   to   token,   "name"   to
clients[viewHolder.adapterPosition]),
                    clientHTTP.createCallback {
                        if (it.getString("message") == "Deleted") {
                            (recyclerView.adapter                as
Adapter).removeAt(viewHolder.adapterPosition)
                        } else {
                            activity?.runOnUiThread {
                Toast.makeText(context,  "Oops.  User  hasn't  been
deleted.", Toast.LENGTH_LONG ).show()
recyclerView.adapter?.notifyItemChanged(viewHolder.adapterPosition)
                            }
                        }
                    })
            }.setNeutralButton("No") { dialog, _ ->
                dialog.dismiss()
            }.setOnDismissListener {
recyclerView.adapter?.notifyItemChanged(viewHolder.adapterPosition)
            }.create().show()
    }

    fun createAllUsersDialog(arr: List<String>) {
        val builder = AlertDialog.Builder(activity!!)
            .setTitle("Select user's name")
        val inflater = this.layoutInflater
        val  dialogView  =  inflater.inflate(R.layout.choose_user,
null)
        builder.setView(dialogView)
        val listView = dialogView.mListView
        val searchBar = dialogView.searchBar
        searchBar.setHint("Seacrh user...")
        val users = arr.toTypedArray()
        val    arrAdapter    =    ArrayAdapter<String>(context!!,
android.R.layout.simple_list_item_1, users)
        listView.adapter = arrAdapter
        searchBar.addTextChangeListener(object : TextWatcher {
            override fun afterTextChanged(p0: Editable?) {
            }
            override fun onTextChanged(p0: CharSequence?, p1: Int,
p2: Int, p3: Int) {
                arrAdapter.filter.filter(p0)

            }
```

```
            override fun beforeTextChanged(p0: CharSequence?, p1:
Int, p2: Int, p3: Int) {
            }
        })

        builder.setOnDismissListener {
            activity?.runOnUiThread {
                view?.findViewById<RecyclerView>(R.id.recyclerView)
                    ?.adapter?.notifyDataSetChanged()
            }
        }
        builder.setNeutralButton("No") { dialogInterface, _ ->
            dialogInterface.dismiss()
        }.create()
        val dialog: AlertDialog = builder.show()
        listView.setOnItemClickListener { _, _, i, _ ->
            var presence = false
            if (arrAdapter.getItem(i) in clients){
                val           handler           =           object:
Handler(Looper.getMainLooper()) {
                    override fun handleMessage(msg: Message) {
                        throw RuntimeException()
                    } }
                AlertDialog.Builder(activity!!)
          .setTitle("Alert")
          .setMessage("This   person   is   already   present   in
training.\n" + arrAdapter.getItem(i) +
           "\nDo you really want to add this abonement again?")
           .setPositiveButton("Yes"){ _, _ ->
handler.sendMessage(handler.obtainMessage())}
           .setNegativeButton("No"){ _, _ ->
                presence = true
handler.sendMessage(handler.obtainMessage())
                }.create().show()

                try {
                    Looper.loop()
                }
                catch (e: RuntimeException){
                    println(e)
                }
            }

            if (!presence){
                val clientHTTP = ClientHTTP(activity, context)
                clientHTTP.sendRequest(
                    "$url/client/find_code",
```

```
                    hashMapOf("token"    to    token,    "name"    to
arrAdapter.getItem(i)!!),
                    clientHTTP.createCallback {
                        when (it.getString("message")) {
                            "Added" -> {
            val newClient = it.getString("name")
            val newAbonInfo = it.getString("abon_info")
            clients.add(newClient)
            abons.add(newAbonInfo)
            activity?.runOnUiThread {
recyclerView.adapter?.notifyItemInserted(clients.size) }
            dialog.dismiss()
                            }
            "Already here" -> activity?.runOnUiThread {
           Toast.makeText(context,          "Already          here",
Toast.LENGTH_LONG).show()
recyclerView.adapter?.notifyDataSetChanged()
            dialog.dismiss()}
          "No user with such code" -> activity?.runOnUiThread {
             Toast.makeText(context,    "Code    wasn't    found",
Toast.LENGTH_LONG).show()}
          "No    active    abonement    for    such    code"    ->
activity?.runOnUiThread {
          val abonements = Json.parse(
          String.serializer().list,
           it.getString("abon_array"))
createAbonDialog(it.getString("code"), abonements)
       dialog.dismiss()}
            }
                    })
            }
        }
    }

    fun abonAlert(choose: String): Boolean{
        var result = false
        val handler = object:  Handler(Looper.getMainLooper()) {
            override fun handleMessage(msg: Message) {
                throw RuntimeException()
            }
        }

        AlertDialog.Builder(activity!!)
            .setTitle("Alert")
            .setMessage("Are    you    sure    that    you    wanna
choose\n$choose")
            .setPositiveButton("Yes"){ _, _ ->
                result = true
```

```
                handler.sendMessage(handler.obtainMessage())
            }
            .setNegativeButton("No"){ _, _ ->
                result = false
                handler.sendMessage(handler.obtainMessage())
            }
            .create().show()
        try {
            Looper.loop()
        }
        catch (e: RuntimeException){
            println(e)
        }
        return result
    }

    fun abonPresence(code: String): Boolean{
        var result = true
        if (!MainActivity.validateCode(code)){ return false }
        var name = ""
        val clientHttp = ClientHTTP(activity, context)
        clientHttp.sendRequest(
                "$url/client/get_name",
                hashMapOf("token" to token, "code" to code),
                clientHttp.createCallback {
                    if (it.getString("name").isNotEmpty()){
                        name = it.getString("name")
                    }
                }
        )
        val handler = object:  Handler(Looper.getMainLooper()) {
            override fun handleMessage(msg: Message) {
                throw RuntimeException()
            }
        }
        if (name.isNotBlank() && name in clients){
            val builder = AlertDialog.Builder(activity!!)
                .setTitle("Alert")
                .setMessage("This  person  is  already  present  in
training.\n" + name +
                 "\nDo you really want to add this abonement again?")
                .setPositiveButton("Yes"){ _, _ ->
                    result = true
                    handler.sendMessage(handler.obtainMessage())
                }
                .setNegativeButton("No"){ _, _ ->
                    result = false
                    handler.sendMessage(handler.obtainMessage())
```

```
                }
            val dialog = builder.create()
            dialog.show()
            try {
                Looper.loop()
            }
            catch (e: RuntimeException){
                println(e)
            }
        }
        return result
    }
}
```

Лістинг файлу CodeFragment.kt

```
package com.ebert_l.sdc_app
import android.content.Intent
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.os.Message
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import androidx.fragment.app.Fragment
import androidx.navigation.Navigation
import com.google.zxing.client.android.Intents
import com.google.zxing.integration.android.IntentIntegrator
import kotlinx.android.synthetic.main.create_user_dialog.*
import kotlinx.android.synthetic.main.create_user_dialog.view.*
import kotlinx.android.synthetic.main.fragment_code.*
import kotlinx.serialization.json.Json
import kotlinx.serialization.list
import kotlinx.serialization.serializer

class CodeFragment : Fragment() {
    val url = MainActivity.url
    val token = MainActivity.token
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val                  qrScannerIntegrator                  =
IntentIntegrator.forSupportFragment(this)
```

```
        qrScannerIntegrator.captureActivity                          =
CustomScannerActivity::class.java
        qrScannerIntegrator.addExtra(Intents.Scan.SCAN_TYPE,
Intents.Scan.MIXED_SCAN)
            .setDesiredBarcodeFormats(IntentIntegrator.QR_CODE)
            .setOrientationLocked(false)
            .setBeepEnabled(false)
            .setPrompt("Put a code inside rectangle to scan it")
        qrScannerIntegrator.initiateScan()
        return  inflater.inflate(R.layout.fragment_code,  container,
false)
    }

    override  fun  onActivityResult(requestCode:  Int,  resultCode:
Int, data: Intent?) {
        try {
            if (data != null) {
            val                        result                        =
IntentIntegrator.parseActivityResult(requestCode, resultCode, data)
            if (result != null) {
                if
(MainActivity.validateCode(result.contents.toString())) {
                    val abonCode = result.contents
                    val clientHTTP = ClientHTTP(activity, context)
                    clientHTTP.sendRequest("$url/client/info",
                        hashMapOf("code"  to  abonCode,  "token"  to
token), clientHTTP.createCallback {
            if (it.getString("message") in arrayOf("Info",
                "No active abonement for such code")){
               activity?.runOnUiThread {
                    info_text.text = it.getString("info")
                sessionHistB.visibility = View.VISIBLE
                 sessionHistB.setOnClickListener {
                       getAbonHistory(abonCode)
                         }
               addAbonementB.visibility = View.VISIBLE
addAbonementB.setOnClickListener { buttonView ->
                  val arr = Json.parse(
String.serializer().list, it.getString("abon_array") )
createAbonDialog(it.getString("code"), arr)
                 }   }
         } else if (it.getString("message") == "No user with such
code"){             activity?.runOnUiThread {
createUserDialog(it.getString("code"))  }
             }})
                } else {
                    activity?.runOnUiThread {
                Toast.makeText(    context,
```

```
      "Code is not valid",Toast.LENGTH_LONG).show()
                    }
                }
            } else {
          super.onActivityResult(requestCode, resultCode, data)
            }
          }
          } catch (e: Exception){
              activity!!.runOnUiThread {
                  val clientHTTP = ClientHTTP(activity, context)
                  clientHTTP.sendRequest("$url/client/log",
hashMapOf("Message" to e.toString()),
                      clientHTTP.createCallback { println(it) })
          Toast.makeText(context, "Error", Toast.LENGTH_SHORT).show()
Navigation.findNavController(view!!).navigate(R.id.action_loginActi
vity2_to_nav_training)
            }
          }
    }
```

Лістинг файлу CustomScannerActivity.java

```
package com.ebert_l.sdc_app;
import android.app.Activity;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.widget.Button;
import androidx.annotation.NonNull;
import com.journeyapps.barcodescanner.CaptureManager;
import com.journeyapps.barcodescanner.DecoratedBarcodeView;
import com.journeyapps.barcodescanner.ViewfinderView;

public class CustomScannerActivity extends Activity implements
        DecoratedBarcodeView.TorchListener {
    private CaptureManager capture;
    private DecoratedBarcodeView barcodeScannerView;
    private Button switchFlashlightButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_custom_scanner);
        barcodeScannerView                                          =
findViewById(R.id.zxing_barcode_scanner);
        barcodeScannerView.setTorchListener(this);
```

```
        switchFlashlightButton                                    =
findViewById(R.id.switch_flashlight);
        ViewfinderView              viewfinderView                =
findViewById(R.id.zxing_viewfinder_view);
        if (!hasFlash()) {
            switchFlashlightButton.setVisibility(View.GONE);
        }
        capture = new CaptureManager(this, barcodeScannerView);
        capture.initializeFromIntent(getIntent(),
savedInstanceState);
        capture.decode();
    }

    @Override
    protected void onResume() {
        super.onResume();
        capture.onResume();
    }

    @Override
    protected void onPause() {
        super.onPause();
        capture.onPause();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        capture.onDestroy();
    }

    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        capture.onSaveInstanceState(outState);
    }

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        return   barcodeScannerView.onKeyDown(keyCode,   event)   ||
super.onKeyDown(keyCode, event);
    }

    private boolean hasFlash() {
        return getApplicationContext().getPackageManager()

.hasSystemFeature(PackageManager.FEATURE_CAMERA_FLASH);
    }
```

```
    public void switchFlashlight(View view) {
        if
(getString(R.string.turn_on_flashlight).equals(switchFlashlightButt
on.getText())) {
            barcodeScannerView.setTorchOn();
        } else {
            barcodeScannerView.setTorchOff();
        }
    }

    @Override
    public void onTorchOn() {
switchFlashlightButton.setText(R.string.turn_off_flashlight);
    }

    @Override
    public void onTorchOff() {
switchFlashlightButton.setText(R.string.turn_on_flashlight);
    }

    @Override
    public    void    onRequestPermissionsResult(int    requestCode,
@NonNull String[] permissions, @NonNull int[] grantResults) {
        capture.onRequestPermissionsResult(requestCode,
permissions, grantResults);
    }
}
```

Лістинг файлу HistoryFragment.kt

```
package com.ebert_l.sdc_app
import android.app.AlertDialog
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.fragment.app.Fragment
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.wdullaer.materialdatetimepicker.date.DatePickerDialog
import kotlinx.android.synthetic.main.fragment_history.*
import kotlinx.android.synthetic.main.fragment_history.view.*
import kotlinx.serialization.json.Json
import kotlinx.serialization.list
import kotlinx.serialization.serializer
import java.text.SimpleDateFormat
```

```
import java.util.*
import java.util.concurrent.CountDownLatch

class HistoryFragment : Fragment() {
    private var clients = listOf<String>()
    private var abons = listOf<String>()
    private var wholeSum = String()
    val url = MainActivity.url
    val token = MainActivity.token

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        val clientHTTP = ClientHTTP(activity, context)
        var datesList = listOf<String>()
        clientHTTP.sendRequest("$url/client/get_lessons_dates",
hashMapOf("token" to token),
            clientHTTP.createCallback {
                if (it.getString("message") == "Have dates") {
      datesList       =       Json.parse(String.serializer().list,
it.getString("dates"))  }
            })
        val dates = mutableListOf<Calendar>()
        val simpleDateFormat = SimpleDateFormat("yy-MM-dd")
        if (datesList.isEmpty()){
            datePickerButton.text                                    =
getString(R.string.no_dates_found)
            datePickerButton.isEnabled = false
            datePickerButton.isClickable = false
        } else{
            for (dateString in datesList) {
                val date = simpleDateFormat.parse(dateString)
                val calendar = Calendar.getInstance()
                calendar.time = date
                dates.add(calendar)
            }
        }
        datePickerButton.setOnClickListener {
            val now = Calendar.getInstance()
            val dpd = DatePickerDialog.newInstance(
                { dpd_v, y, m, d ->
                    val date = "$y-${String.format("%02d", m + 1)}-
${String.format("%02d", d)}"
                    datePickerButton.text = date
                    var groups = listOf<String>()
                    val countDownLatch = CountDownLatch(1)
clientHTTP.sendRequest("$url/client/get_history",
          hashMapOf("token" to token, "date" to date),
          clientHTTP.createCallback {
```

```
            when (it.getString("message")) {
            "Old lesson" -> {
            activity?.runOnUiThread {
            view.lessonConstLayout.visibility = View.VISIBLE
            clients = Json.parse(
                 String.serializer().list, it.getString("clients"))
            abons = Json.parse( String.serializer().list,
                     it.getString("abons"))
            wholeSum = it.getString("whole_sum")
            instrView2.text = getString( R.string.instructors,
it.getString("instructors"))
            groupView2.text       =       getString(R.string.group,
it.getString("group"))
            sumView.text = getString(R.string.sum, wholeSum)
            recyclerView.layoutManager                                =
LinearLayoutManager(context)
            recyclerView.adapter = Adapter(clients, abons)
recyclerView.setHasFixedSize(true) }
      countDownLatch.countDown()
      }
    "More than one group" -> {
        groups     =          Json.parse(String.serializer().list,
it.getString("groups"))
        countDownLatch.countDown()
                                  }
                              }
                          })
                    countDownLatch.await()
                    if (groups.isNotEmpty()) {
                        AlertDialog.Builder(context)
                            .setTitle("Choose group")
               .setItems(groups.toTypedArray()) { _, which ->
clientHTTP.sendRequest("$url/client/get_history",
                      hashMapOf("token" to token, "date" to date,
                        "sdc_group" to groups[which] ),
                  clientHTTP.createCallback {
                    when (it.getString("message")) {
                            "Old lesson" -> {
activity?.runOnUiThread {
view.lessonConstLayout.visibility = View.VISIBLE
       clients     =     Json.parse(     String.serializer().list,
it.getString("clients")       )
       abons         =         Json.parse(String.serializer().list,
it.getString("abons") )
       wholeSum = it.getString("whole_sum")
       instrView2.text      =      getString(R.string.instructors,
it.getString("instructors"))
```

```
         groupView2.text      =      getString(      R.string.group,
it.getString("group") )
         sumView.text = getString(R.string.sum, wholeSum)

recyclerView.layoutManager =  LinearLayoutManager(context)
recyclerView.adapter = Adapter(clients, abons)
recyclerView.setHasFixedSize(true)
                        }       }   }
      })   }.create().show()
                    }
                }, now.get(Calendar.YEAR), now.get(Calendar.MONTH),
now.get(Calendar.DAY_OF_MONTH)
            )
            dpd.selectableDays = dates.toTypedArray()
            dpd.show(fragmentManager!!, "Datepickerdialog")
        }
        super.onViewCreated(view, savedInstanceState)
    }
```

Лістинг файлу LoginActivity.kt

```
package com.ebert_l.sdc_app
import android.content.Context
import android.os.Bundle
import android.text.Editable
import android.text.TextWatcher
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.EditText
import android.widget.TextView
import android.widget.Toast
import androidx.fragment.app.Fragment
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProviders
import androidx.navigation.Navigation
import kotlinx.android.synthetic.main.activity_login.*

class LoginActivity : Fragment(){
    private lateinit var loginViewModel: LoginViewModel
    private val KEY_USERNAME = "prefUsernameKey"
    private val PREFERENCE_FILE_KEY = "myAppPreference"

    override fun onCreateView(inflater: LayoutInflater, container:
ViewGroup?, savedInstanceState: Bundle?): View? {
        val   view   =   inflater.inflate(R.layout.activity_login,
container, false)
```

```
        val                    sharedPreferences                    =
activity?.getSharedPreferences(PREFERENCE_FILE_KEY,
Context.MODE_PRIVATE)
        val  username  =  sharedPreferences?.getString(KEY_USERNAME,
"")

view.findViewById<EditText>(R.id.loginEdit).setText(username,
TextView.BufferType.EDITABLE)
        return view
    }

    override  fun  onViewCreated(view:  View,  savedInstanceState:
Bundle?) {
        loginViewModel                                              =
ViewModelProviders.of(this).get(LoginViewModel::class.java)
        loginViewModel.client        =        (activity        as
MainActivity).getSelfCertClient()
        loginViewModel.loginFormState.observe(this, Observer {
            val loginState = it ?: return@Observer
            loginButton.isEnabled = loginState.isDataValid
            if (loginState.usernameError != null) {
      loginEdit.error = getString(loginState.usernameError)
            }
            if (loginState.passwordError != null) {
      passEdit.error = getString(loginState.passwordError)
            } })

        loginEdit.afterTextChanged {
            loginViewModel.loginDataChanged(
                loginEdit.text.toString(),
                passEdit.text.toString()
            )
        }
        passEdit.apply {
            afterTextChanged {
                loginViewModel.loginDataChanged(
                    loginEdit.text.toString(),
                    passEdit.text.toString()
                )
            }

           loginButton.setOnClickListener {
    val  loginRes  =  loginViewModel.login(loginEdit.text.toString(),
passEdit.text.toString())                if (loginRes){
    val                          sharedPref                          =
activity?.getSharedPreferences(PREFERENCE_FILE_KEY,
Context.MODE_PRIVATE)
```

```
        sharedPref?.edit()?.putString(KEY_USERNAME,
loginEdit.text.toString())?.apply()
        MainActivity.token = loginViewModel.token_
Navigation.findNavController(view).navigate(R.id.action_loginActivi
ty2_to_nav_training)
                } else{
                    Toast.makeText(context, loginViewModel.message,
Toast.LENGTH_LONG).show()
                }           }
        }
        super.onViewCreated(view, savedInstanceState)
    }
}

fun EditText.afterTextChanged(afterTextChanged: (String) -> Unit) {
    this.addTextChangedListener(object : TextWatcher {
        override fun afterTextChanged(editable: Editable?) {
            afterTextChanged.invoke(editable.toString())
        }
        override fun beforeTextChanged(s: CharSequence, start: Int,
count: Int, after: Int) {}
        override fun onTextChanged(s: CharSequence, start: Int,
before: Int, count: Int) {}
    })

}
```

Лістинг файлу LoginViewModel.kt

```
package com.ebert_l.sdc_app
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import okhttp3.*
import org.json.JSONObject
import java.io.IOException
import java.util.concurrent.CountDownLatch

class LoginViewModel : ViewModel() {
    private val _loginForm = MutableLiveData<LoginFormState>()
    val loginFormState: LiveData<LoginFormState> = _loginForm
    var token_: String = ""
    val url = MainActivity.url
    var message: String = "Authorization failed"
    lateinit var client: OkHttpClient

    fun login(phone: String, password: String) : Boolean{
        var res = false
```

```
        val    requestBody    =    FormBody.Builder().add("phone",
phone).add("password", password).build()
        val                          request                          =
Request.Builder().url("$url/client/reg").post(requestBody).build()
        val countDownLatch = CountDownLatch(1)
        client.newCall(request).enqueue(object : Callback {
            override fun onFailure(call: Call, e: IOException) {
                println(e)
                message = e.toString()
                countDownLatch.countDown()
            }

      override fun onResponse(call: Call, response: Response) {
                val body = response.body()!!.string()
                try {
                    val jsonObject = JSONObject(body)
          if (jsonObject.getString("message") == "Success"){
                        if (jsonObject.getBoolean("result")){
                            token_ = jsonObject.getString("token")
                            res = true
            message = jsonObject.getString("message")
                        }
       } else if (jsonObject.getString("message") == "Failure"){
                        res = false
                        token_ = ""
                        message = jsonObject.getString("error")
                    }
                } catch (e: Exception) {
                    message = e.toString()
                }
                countDownLatch.countDown()
            }
        })
        countDownLatch.await()
        return res
    }

    fun loginDataChanged(username: String, password: String) {
        if (!isUserNameValid(username)) {
            _loginForm.value   =   LoginFormState(usernameError   =
R.string.invalid_username)
        } else if (!isPasswordValid(password)) {
            _loginForm.value   =   LoginFormState(passwordError   =
R.string.invalid_password)
        } else {
            _loginForm.value = LoginFormState(isDataValid = true)
        }
    }
```

```
    private fun isUserNameValid(username: String): Boolean {
        return username.length >= 5
    }

    private fun isPasswordValid(password: String): Boolean {
        return password.length > 5
    }

    data class LoginFormState(
        val usernameError: Int? = null,
        val passwordError: Int? = null,
        val isDataValid: Boolean = false
    )
}
```

Лістинг файлу SwipeToDeleteCallback.kt

```
package com.ebert_l.sdc_app
import android.content.Context
import android.graphics.Color
import android.graphics.Paint
import android.graphics.PorterDuff
import android.graphics.PorterDuffXfermode
import android.graphics.drawable.ColorDrawable
import androidx.core.content.ContextCompat
import androidx.recyclerview.widget.ItemTouchHelper
import androidx.recyclerview.widget.RecyclerView
import android.graphics.Canvas

abstract class SwipeToDeleteCallback(context: Context):
ItemTouchHelper.SimpleCallback(0, ItemTouchHelper.LEFT) {
    private val deleteIcon = ContextCompat.getDrawable(context,
R.drawable.ic_delete_white_24dp)
    private val intrinsicWidth = deleteIcon!!.intrinsicWidth
    private val intrinsicHeight = deleteIcon!!.intrinsicHeight
    private val background = ColorDrawable()
    private val backgroundColor = Color.parseColor("#f44336")
    private val clearPaint = Paint().apply { xfermode =
PorterDuffXfermode(PorterDuff.Mode.CLEAR) }

    override fun onMove(
        recyclerView: RecyclerView,
        viewHolder: RecyclerView.ViewHolder,
        target: RecyclerView.ViewHolder
    ): Boolean {
        return false
```

```
    }

    override fun onChildDraw(
        c: Canvas,
        recyclerView: RecyclerView,
        viewHolder: RecyclerView.ViewHolder,
        dX: Float,
        dY: Float,
        actionState: Int,
        isCurrentlyActive: Boolean
    ) {
        val itemView = viewHolder.itemView
        val itemHeight = itemView.bottom - itemView.top
        val isCanceled = dX == 0f && !isCurrentlyActive

        if (isCanceled) {
            clearCanvas(c,        itemView.right        +        dX,
itemView.top.toFloat(),                    itemView.right.toFloat(),
itemView.bottom.toFloat())
            super.onChildDraw(c, recyclerView, viewHolder, dX, dY,
actionState, isCurrentlyActive)
            return
        }
        background.color = backgroundColor
        background.setBounds(itemView.right      +      dX.toInt(),
itemView.top, itemView.right, itemView.bottom)
        background.draw(c)
        val   deleteIconTop   =   itemView.top   +   (itemHeight   -
intrinsicHeight) / 2
        val deleteIconMargin = (itemHeight - intrinsicHeight)  + 15
        val  deleteIconLeft  =  itemView.right  -  deleteIconMargin  -
intrinsicWidth
        val deleteIconRight = itemView.right - deleteIconMargin
        val deleteIconBottom = deleteIconTop + intrinsicHeight

        deleteIcon!!.setBounds(deleteIconLeft,       deleteIconTop,
deleteIconRight, deleteIconBottom)
        deleteIcon.draw(c)

        super.onChildDraw(c,  recyclerView,  viewHolder,  dX,  dY,
actionState, isCurrentlyActive)
    }

    private fun clearCanvas(c: Canvas?, left: Float, top: Float,
right: Float, bottom: Float) {
        c?.drawRect(left, top, right, bottom, clearPaint)
    }}
```

Лістинг файлу UserFragment.kt

```
package com.ebert_l.sdc_app
import android.app.AlertDialog
import android.content.Intent
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.os.Message
import android.text.Editable
import android.text.TextWatcher
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import android.widget.ListView
import android.widget.Toast
import androidx.navigation.Navigation
import com.google.zxing.client.android.Intents
import com.google.zxing.integration.android.IntentIntegrator
import com.mancj.materialsearchbar.MaterialSearchBar
import kotlinx.android.synthetic.main.create_user_dialog.*
import kotlinx.android.synthetic.main.create_user_dialog.view.*
import kotlinx.serialization.json.Json
import kotlinx.serialization.list
import kotlinx.serialization.serializer

class UserFragment : Fragment() {
    var usersList = listOf<String>()
    var row = String()
    val url = MainActivity.url
    val token = MainActivity.token

    override fun onCreate(savedInstanceState: Bundle?) {
        val clientHTTP = ClientHTTP(activity, context)
        clientHTTP.sendRequest("$url/client/get_all_users_info",
hashMapOf("token" to token),
            clientHTTP.createCallback {
                if (it.getString("message") == "All users info"){
                    usersList                                          =
Json.parse(String.serializer().list, it.getString("users"))
                }
            })
        super.onCreate(savedInstanceState)
    }

    override fun onCreateView(
```

```
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val    view    =    inflater.inflate(R.layout.fragment_user,
container, false)
        val users = usersList.toTypedArray()
        val     arrAdapter     =     ArrayAdapter<String>(context!!,
android.R.layout.simple_list_item_1, users)
        val listView = view.findViewById<ListView>(R.id.listView)
        listView.adapter = arrAdapter
        val                          searchBar                          =
view.findViewById<MaterialSearchBar>(R.id.searchUserBar)
        searchBar.setHint("Enter name...")
        searchBar.addTextChangeListener(object : TextWatcher {
            override fun afterTextChanged(p0: Editable?) {
            }

            override fun onTextChanged(p0: CharSequence?, p1: Int,
p2: Int, p3: Int) {
                arrAdapter.filter.filter(p0)
            }
            override fun beforeTextChanged(p0: CharSequence?, p1:
Int, p2: Int, p3: Int) {
            }
        })

     listView.setOnItemClickListener { adapterView, view_, i, l ->
            val actions = listOf("Change QR code by scanning",
"Change user data", "Add abonement",
                "\t\tCancel")
            AlertDialog.Builder(activity)
                .setTitle("Choose action")
                .setItems(actions.toTypedArray()){ _, which ->
                    when (which){
                        0 -> {
            row = arrAdapter.getItem(i)!!.toString()
                            val        qrScannerIntegrator        =
IntentIntegrator.forSupportFragment(this)
                            qrScannerIntegrator.captureActivity   =
CustomScannerActivity::class.java
qrScannerIntegrator.addExtra(Intents.Scan.SCAN_TYPE,
Intents.Scan.MIXED_SCAN).setDesiredBarcodeFormats(IntentIntegrator.
QR_CODE).setOrientationLocked(false).setBeepEnabled(false)
       .setPrompt("Put a code inside rectangle to scan it")
                            qrScannerIntegrator.initiateScan()
                        }
                        1 -> {
row = arrAdapter.getItem(i)!!.toString()
```

```
                            editUserData(row)
                        }
                        2 -> {
                            row                                    =
arrAdapter.getItem(i)!!.toString()
                            newAbon(row)
                        }
                    }
                }.create().show()
        }
        return view
    }

    override fun onActivityResult(requestCode: Int, resultCode:
Int, data: Intent?) {
        try {
            if (data != null) {
                val                    result                    =
IntentIntegrator.parseActivityResult(requestCode, resultCode, data)

                if (result != null) {
                    if
(MainActivity.validateCode(result.contents.toString())
                        && row.isNotEmpty()) {
                        val   clientHTTP   =   ClientHTTP(activity,
context)

clientHTTP.sendRequest("$url/client/change_user_code",
   hashMapOf("token"        to        token,        "code"        to
result.contents.toString(),"row" to row), clientHTTP.createCallback
{
when(it.getString("message")){
         "Problems occurred" -> {
          activity?.runOnUiThread {
        Toast.makeText(      context,      "Problems      occurred",
Toast.LENGTH_LONG).show()
                                        }
                                    }
      "Code changed" -> {
          activity?.runOnUiThread {
        Toast.makeText(context,           "Code           changed",
Toast.LENGTH_LONG).show()
                                        }
                                    }
                                }
                            })

                    } else {
```

```
                activity?.runOnUiThread {
            Toast.makeText(context,          "Code          is          not
valid",Toast.LENGTH_LONG).show()
                            }
                        }
                    } else {
                        super.onActivityResult(requestCode, resultCode,
data)  }
                }
            } catch (e: Exception){
                activity!!.runOnUiThread {
                    val clientHTTP = ClientHTTP(activity, context)
                    clientHTTP.sendRequest("$url/client/log",
hashMapOf("Message" to e.toString()),
                        clientHTTP.createCallback { println(it) })
                    Toast.makeText(context,                      "Error",
Toast.LENGTH_SHORT).show()
Navigation.findNavController(view!!).navigate(R.id.action_loginActi
vity2_to_nav_training)
                }
            }
        }
```

Лістинг файлу navgraph.xml

```
<?xml version="1.0" encoding="utf-8"?>
<navigation
xmlns:android="http://schemas.android.com/apk/res/android"
            xmlns:app="http://schemas.android.com/apk/res-auto"
            xmlns:tools="http://schemas.android.com/tools"
        app:startDestination="@id/loginActivity">

    <fragment                          android:id="@+id/nav_training"
android:name="com.ebert_l.sdc_app.TrainFragment1"
              android:label="train_fragment1"
tools:layout="@layout/train_fragment1">
        <action
android:id="@+id/action_nav_training_to_trainFragment2"
app:destination="@id/trainFragment2"/>
        <action
android:id="@+id/action_nav_training_to_trainFragment3"
app:destination="@id/trainFragment3"
                app:popUpTo="@+id/nav_training"/>
        <action

android:id="@+id/action_nav_training_to_historyFragment"
                app:destination="@id/historyFragment" />
```

```
        <action
android:id="@+id/action_nav_training_to_reportFragment"
                app:destination="@id/reportFragment" />
        <action
android:id="@+id/action_nav_training_to_userFragment"
                app:destination="@id/userFragment" />
    </fragment>

    <fragment                       android:id="@+id/trainFragment2"
android:name="com.ebert_l.sdc_app.TrainFragment2"
             android:label="train_fragment2"
tools:layout="@layout/train_fragment2">
        <action
android:id="@+id/action_trainFragment2_to_trainFragment3"
app:destination="@id/trainFragment3"
                app:popUpTo="@+id/nav_training"/>
    </fragment>
    <fragment                               android:id="@+id/nav_code"
android:name="com.ebert_l.sdc_app.CodeFragment"
             android:label="fragment_code"
tools:layout="@layout/fragment_code"/>
    <fragment                       android:id="@+id/trainFragment3"
android:name="com.ebert_l.sdc_app.TrainFragment3"
             android:label="train_fragment3"
tools:layout="@layout/train_fragment3"/>
    <fragment android:name="com.ebert_l.sdc_app.LoginActivity"
             android:label="LoginActivity"
android:id="@+id/loginActivity"
tools:layout="@layout/activity_login">
        <action
android:id="@+id/action_loginActivity2_to_nav_training"
app:destination="@id/nav_training"
                app:popUpToInclusive="false"
app:popEnterAnim="@anim/nav_default_pop_enter_anim"
                app:popExitAnim="@anim/nav_default_pop_exit_anim"/>
    </fragment>
    <action
            android:id="@+id/action_global_loginActivity"
            app:destination="@id/loginActivity"
            app:popUpTo="@id/loginActivity" />

    <fragment
            android:id="@+id/historyFragment"
            android:name="com.ebert_l.sdc_app.HistoryFragment"
            android:label="fragment_history"
            tools:layout="@layout/fragment_history" />
    <fragment
            android:id="@+id/reportFragment"
```

```
            android:name="com.ebert_l.sdc_app.ReportFragment"
            android:label="fragment_report"
            tools:layout="@layout/fragment_report" />
    <fragment
            android:id="@+id/userFragment"
            android:name="com.ebert_l.sdc_app.UserFragment"
            android:label="fragment_user"
            tools:layout="@layout/fragment_user" />
</navigation>
```