Федеральное государственное автономное

образовательное учреждение

Высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт космических и информационных технологий |
| институт |
| Информатика |
| кафедра |

**ОТЧЁТ О ПРАКТИЧЕСКОЙ РАБОТЕ №3**

|  |
| --- |
| Использование локальной базы данных |
| тема |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Преподаватель | |  |  |  | А. С. Кузнецов |
|  | |  | подпись, дата |  | инициалы, фамилия |
| Студент | КИ21-17/1Б, 032157401 |  |  |  | Э.О. Варков |
|  | номер группы, зачетной книжки |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2025

Задание

Подключить к приложению из практической работы №2 локальную базу данных (особых требований к базе данных нет, она может быть как реляционной так и нереляционной, например: SQLite или Realm).

Написать методы, которые будут реализовывать CRUD-операции для каких-либо целей (первый пункт в описании требований к практической работе №1). "Подключить" эти методы к пользовательскому интерфейсу (например, кнопка "Сохранить заметку" должна сохранять заметку).

Составить инструкцию по сборке apk-файла вашего приложения средствами интерфейса командной строки.

Сформировать и загрузить архив установленного формата с исходным кодом проекта и инструкцией по сборке вашего приложения или привести ссылку (-ки) на открытый репозиторий кода и инструкцией по сборке, а также (необязательно) на собранный apk-файл.

Описание приложения

Полочка — это Android-приложение, представляющее собой доску объявлений для продажи и передачи книг. Пользователи могут размещать объявления, искать книги по названию и находить доступные предложения поблизости. Для удобного указания местоположения при создании объявления используется карта. Приложение включает клиентскую и серверную части, обеспечивая хранение и доступ к объявлениям с разных устройств.

Ход работы

* 1. **Создание сервера**
     1. **Необходимость в сервере**

Так как приложение представляет собой доску объявлений, данные должны сохраняться и быть доступны для всех пользователей. Использование локальной базы данных не подходит, поскольку в этом случае объявления хранились бы только на одном устройстве. Сервер обеспечивает централизованное хранение объявлений, и доступ к ним с разных устройств.

* + 1. **Выбор инструментов**

Для серверной части был выбран FastAPI, так как он прост в освоении и удобен для разработки API. В качестве базы данных используется SQLite, поскольку это легковесное решение, не требующее отдельного сервера, что упрощает разработку. Такой выбор обусловлен тем, что основное внимание уделяется мобильной разработке на Java, а не серверной части, поэтому использовались более простые инструменты.

* + 1. **Хранение объявлений**

Для хранения объявлений используется база данных SQLite. Каждое объявление представляет собой запись в таблице books, содержащей такие поля, как название книги, автор, описание, местоположение, координаты, контактная информация пользователя и другие данные. Для добавления, удаления и получения объявлений используется REST API, реализованный с помощью FastAPI.

Листинг 1 — Добавление книги

@app.post("/books/")

async def create\_book(book: BookRequest, session: AsyncSession = Depends(get\_session)):

title = capitalize\_first\_letter(book.title)

author = capitalize\_first\_letter(book.author)

username = capitalize\_first\_letter(book.username)

city = capitalize\_first\_letter(shorten\_city\_name(book.city))

district = capitalize\_first\_letter(book.district)

new\_book = BookModel(

title=title,

author=author,

username=username,

city=city,

phone\_number=book.phone\_number,

description=book.description,

longitude=book.longitude,

latitude=book.latitude,

district=district,

)

session.add(new\_book)

await session.commit()

logger.info(f"Добавлена книга: {new\_book.title}")

* + 1. **Хранение изображений**

Для хранения изображений книг используется отдельная папка на сервере, куда загружаются файлы с изображениями. Каждый файл привязывается к конкретному объявлению через идентификатор книги. Изображение сохраняется с уникальным именем, которое формируется на основе идентификатора книги. При загрузке изображения на сервер, оно сохраняется в директории imgs. Для этого используется метод upload\_file, который принимает файл изображения и идентификатор книги.

Листинг 1 — Добавление изображений

@app.post("/upload")

async def upload\_file(file: UploadFile = File(...), book\_id: str = Form(...)):

images\_dir = os.path.join(BASE\_DIR, "imgs")

os.makedirs(images\_dir, exist\_ok=True)

file\_extension = os.path.splitext(file.filename)[1]

file\_path = os.path.join(images\_dir, f"{book\_id}{file\_extension}")

# Явное удаление файла, если он существует

if os.path.exists(file\_path):

os.remove(file\_path)

with open(file\_path, "wb") as f:

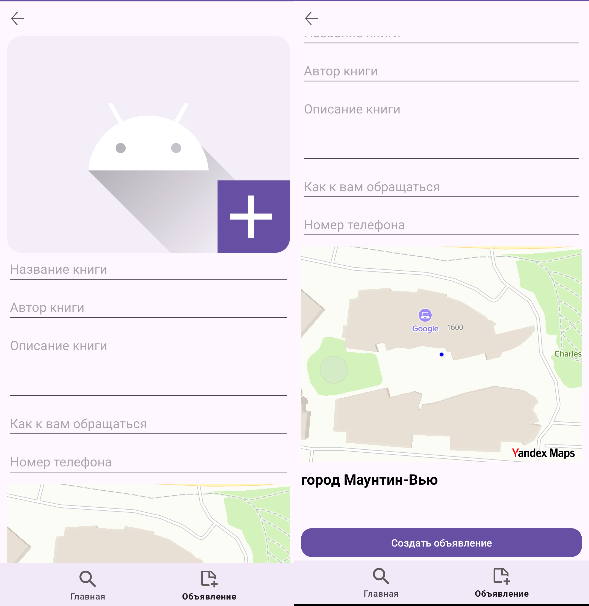
f.write(await file.read())

return {"message": "Файл успешно загружен"}

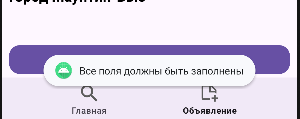
* 1. **Отправка на сервер**

Шаги реализации:

Сбор данных: На первом этапе данные, введённые пользователем в форме, собираются из соответствующих полей ввода.

Рисунок 1 — Форма ввода

Проверка на корректность ввода: Прежде чем отправить данные на сервер, выполняется проверка, чтобы убедиться, что все обязательные поля заполнены. В случае ошибки пользователю выводится сообщение.

Рисунок 2 — Сообщение об ошибке

Создание JSON-объекта: После проверки, собранные данные помещаются в объект JSONObject, который формирует тело запроса для отправки на сервер.

Отправка данных: Далее, с помощью объекта ServerCommunicator происходит отправка данных на сервер методом POST. Это выполняется с использованием библиотеки OkHttp, которая отправляет JSON-объект на указанный URL с помощью асинхронного запроса.

Листинг 2 — Отправка

JSONObject payload = new JSONObject();  
payload.put("title", title);  
payload.put("author", author);  
payload.put("description", description);  
payload.put("city", city);  
payload.put("district", district);  
payload.put("username", userName);  
payload.put("phone\_number", userNumber);  
payload.put("latitude", latitude);  
payload.put("longitude", longitude);

serverCommunicator.sendBook(payload, new Callback() {

@Override

public void onFailure(Call call, IOException e) {

handleServerError(e);

}

@Override

public void onResponse(Call call, Response response) throws IOException {

handleServerResponse(response);

}

});

...

public class ServerCommunicator {

private final String serverUrl;

private final OkHttpClient client;

public ServerCommunicator(String serverUrl) {

this.serverUrl = serverUrl;

this.client = new OkHttpClient();

}

public void sendBook(JSONObject payload, Callback callback) {

RequestBody body = RequestBody.create(payload.toString(), MediaType.get("application/json; charset=utf-8"));

Request request = new Request.Builder()

.url(serverUrl + "/books")

.post(body)

.build();

client.newCall(request).enqueue(callback);

}

}

...

* 1. **Получение с сервера**

Шаги реализации:

Отправка GET-запроса: С помощью библиотеки OkHttp отправляется GET-запрос на URL /books, который возвращает список книг в формате JSON.

Листинг 3 — Отправка запроса

OkHttpClient client = new OkHttpClient();

Request request = new Request.Builder()

.url(SERVER\_URL + "/books") // URL API

.build();

Обработка ответа: В случае успешного ответа от сервера данные разбираются из JSON-формата, и извлекается массив книг. Для каждой книги создается объект модели ItemModel, который затем добавляется в список.

Листинг 4 — Обработка ответа

String responseBody = response.body().string();

JSONObject jsonResponse = new JSONObject(responseBody);

JSONArray booksArray = jsonResponse.getJSONArray("books");

ArrayList<ItemModel> newItems = new ArrayList<>();

for (int i = 0; i < booksArray.length(); i++) {

JSONObject bookObject = booksArray.getJSONObject(i);

int id = bookObject.getInt("id");

String title = bookObject.getString("title");

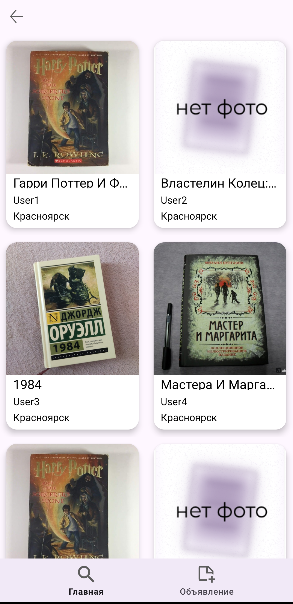
String author = bookObject.getString("author");

// Извлечение других данных...

newItems.add(new ItemModel(id, title, author, /\* другие параметры \*/));

}

Обновление интерфейса: После того, как все книги были получены и обработаны, обновляется пользовательский интерфейс с новыми данными.

Рисунок 3 — Обновление интерфейса

* 1. **Добавление карты**

Для отображения карты в приложении использовался Yandex MapKit API.

овательский интерфейс с новыми данными.

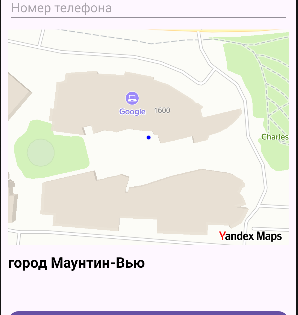
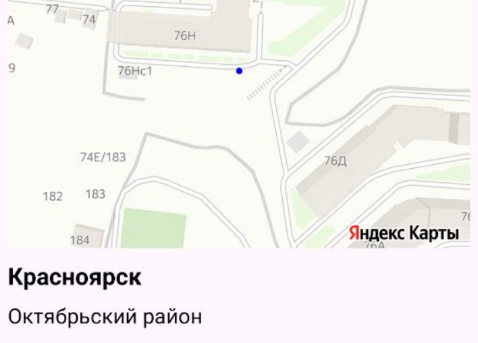


Рисунок 4 — Карта

* 1. **Добавление геокодера**

Для преобразования координат, получаемых с карты, в понятные пользователю топонимы использовался API Геокодера Яндекса. Геокодер позволяет по переданным координатам определять адреса или другие географические объекты, предоставляя информацию в удобной текстовой форме.

Рисунок 4 — Определение топонимов по координатам

* 1. **Дальнейшая разработка**

В дальнейшем планируется добавление следующих функциональных возможностей:

Авторизация: Введение системы регистрации и входа в приложение для пользователей, что позволит персонализировать опыт и хранить данные о пользователях.

Избранное: Возможность добавлять объявления в избранное для удобного доступа и просмотра.

Удаление своих объявлений: Реализация функции удаления объявлений, созданных пользователями.

Поиск по названиям: Добавление функции поиска книг по названию, что обеспечит удобный поиск информации среди большого числа объявлений.

1. Вывод

В процессе разработки приложения была реализована основная функциональность, включающая отправку и получение данных с сервера, отображение карты и использование геокодера для преобразования координат в топонимы.