

09.02.07 ПР-312

## **ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

**ПП.01.01 Разработка мобильных приложений**

**ПМ.01 Разработка модулей программного обеспечения**

**Практикант**

**Филатов Н.А.**

**Руководители практики**

**Домбровский Н.С.  
Смирнова Е.Е..**

<b>Оглавление</b>	
<b>Задание на разработку .....</b>	<b>3</b>
<b>Анализ индивидуального задания .....</b>	<b>4</b>
<b>Проектирование дизайна приложения .....</b>	<b>6</b>
<b>Описание данных приложения .....</b>	<b>8</b>
<b>Описание дизайна приложения.....</b>	<b>10</b>
<b>Реализация функций приложения.....</b>	<b>20</b>
<b>Заключение .....</b>	<b>37</b>
<b>Список литературы.....</b>	<b>38</b>

## Задание на разработку

### Цель:

Разработать мобильное приложение для поиска и бронирования отелей с функциями управления профилем, избранным и чатом с менеджерами «Путешествия: *Travenor*».

### Основные задачи:

Модуль 1: Разработка технического задания

- Описание функциональных требований
- Проектирование структуры данных
- Создание макета приложения

2. Модуль 2: Разработка серверной части

3. Модуль 3: Верстка мобильного приложения

4. Модуль 4: Разработка функционала мобильного приложения

5. Модуль 5: Тестирование приложения

6. Модуль 6: Документирование результатов разработки

### Общие требования:

1. Использовать систему контроля версий Git, ежедневно сохранять прогресс.

2. Проект должен быть структурирован: исходные файлы в соответствующих каталогах.

3. Корректная обработка серверных ошибок и отсутствие соединения — пользователю показывать диалоговое окно с ошибкой.

## **Анализ индивидуального задания**

Разработка мобильного приложения для бронирования отелей представляет собой комплексную задачу, требующую интеграции нескольких ключевых функциональных модулей, включая управление пользователями, бронированиями, чаты с менеджерами и фильтрацию отелей.

### **Модуль 1. Разработка технического задания**

На начальном этапе определяется концепция приложения, формулируются цели и задачи, разрабатывается структура данных и создается детализированный дизайн-макет в Figma.

### **Модуль 2. Разработка серверной части**

В качестве серверной части приложения используется Supabase, обеспечивающий:

- Хранение данных (отели, бронирования, пользователи).
- Аутентификацию (email + пароль, восстановление через OTP).
- Политики безопасности (ограничение доступа к данным на основе ролей).
- Хранение медиафайлов (фотографии отелей, аватары пользователей).

### **Модуль 3. Верстка мобильного приложения**

Интерфейс приложения разрабатывается в соответствии с макетом в Figma, с адаптацией под различные размеры экранов. Основные элементы:

#### **Навигация:**

- Нижнее меню для быстрого перехода между разделами (главная, поиск, бронирования, чаты, профиль).

#### **Экраны:**

- Onboarding, авторизация, восстановление пароля.
- Поиск отелей с фильтрами.
- Детальная страница отеля.
- Чат с менеджером.

## **Модуль 4. Разработка функционала**

Финальный этап включает реализацию всех запланированных функций:

Аутентификация и безопасность:

- Вход по email и паролю.
- Восстановление пароля через ОТР.
- Локальное хранение сессии.

Работа с отелями:

- Поиск и фильтрация (Название, количество звезд, поиск по контексту опимания).
- Добавление в избранное.
- Бронирование с выбором дат.

Чат с менеджером:

- Отправка и получение сообщений.

Локализация:

- Поддержка русского и английского языков.

## **Модуль 5. Тестирование**

Тестирование навигации, форм, сценариев ошибок, работы с сервером и обработкой исключений.

## **Модуль 6. Документация**

Создание отчёта.

# Проектирование дизайна приложения

## Основные принципы используемые при создании дизайна:

1. Минимализм и удобство — лаконичный дизайн без лишних элементов, упор на удобство навигации.
2. Адаптивность — макеты корректно отображаются на экранах разных размеров.
3. Консистентность — единый стиль цветов, шрифтов и иконок для всех экранов.
4. Доступность — соблюдение контрастности, читаемости текста и удобных размеров интерактивных элементов.

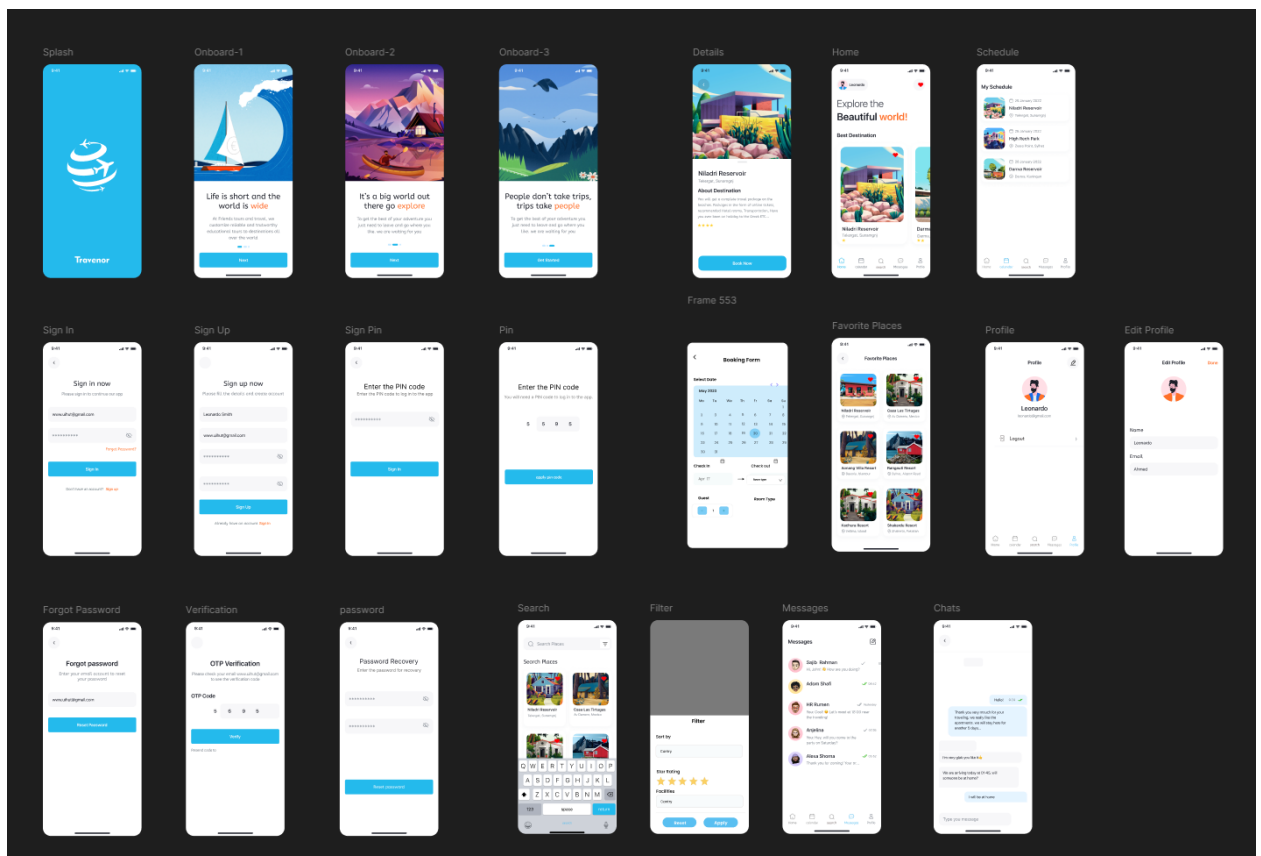


Рисунок 1 Макет Figma

## Описание важных элементов:

- Экран приветствия: Простой и понятный интерфейс с кнопками для входа и регистрации.

- Экран поиска отелей: Фильтры по контексту, количеству звезд и местоположению.
- Экран чата: Удобный интерфейс для общения с менеджерами отелей.
- Экран избранного: отображает элементы которые выбрал клиент

## Описание данных приложения

База данных реализована на Supabase (PostgreSQL) и включает несколько взаимосвязанных таблиц (рис. 2), оптимизированных для быстрого доступа, масштабируемости и целостности данных.

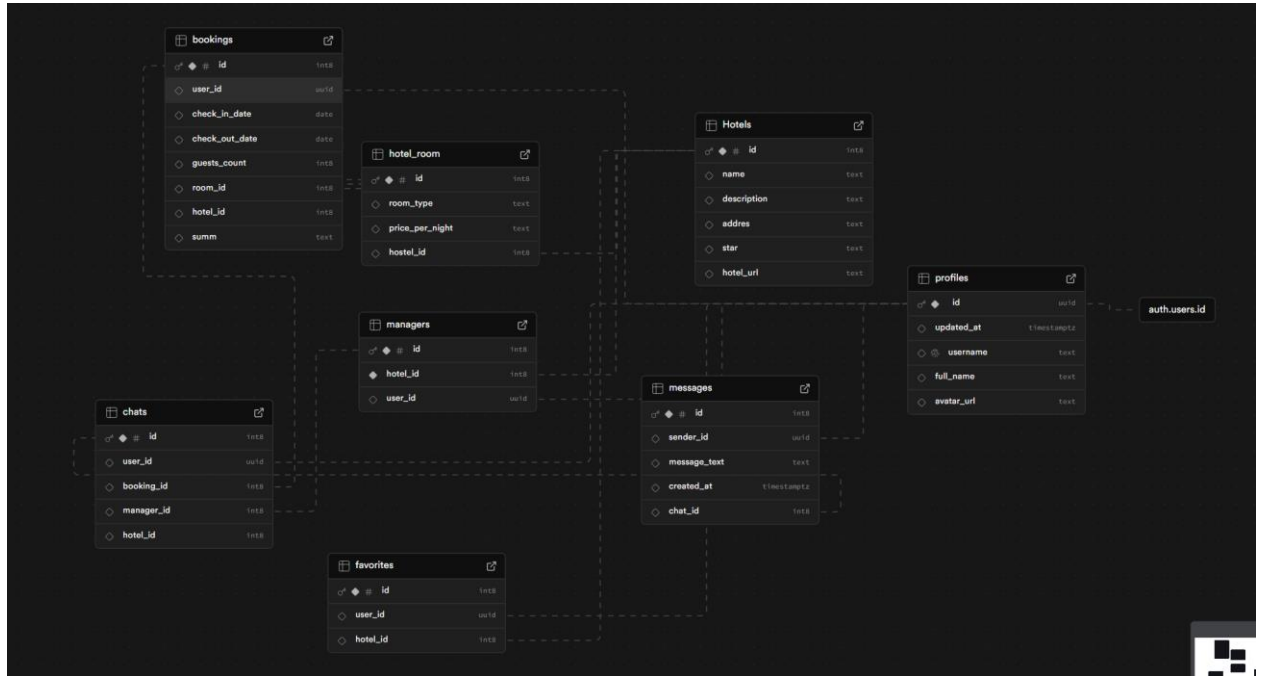


Рисунок 2 Схема БД для приложения

Users (id, full\_name, email, avatar\_url)

Chats (id, user\_id, manager\_id, hotel\_id, booking\_id)

Messages (id, chat\_id, sender\_id, text, created\_at)

Favorites (id, user\_id, hotel\_id)

Hotels (id, name, address, description, stars, hotel\_url)

Bookings (id, user\_id, hotel\_id, check\_in\_date, check\_out\_date, guests\_count, hotel\_id, ssum)

Hotel\_room (id, room\_type, prise\_per\_night, hostel\_id)

1. Пользователи ↔ Чаты

chats.user\_id → users.id

chats.manager\_id → users.id

2. Сообщения ↔ Чаты/Пользователи

messages.chat\_id → chats.id



messages.sender\_id → users.id

3. Избранное ↔ Пользователи/Отели

favorites.user\_id → users.id

favorites.hotel\_id → hotels.id

4. Бронирования ↔ Пользователи/Отели

bookings.user\_id → users.id

bookings.hotel\_id → hotels.id

5. Чаты ↔ Отели/Бронирования

chats.hotel\_id → hotels.id

chats.booking\_id → bookings.id

6. Номера отелей ↔ Отели

hotel\_room.hotel\_id → hotels.id

### **Политики безопасности:**

Была использована политик предложенная самим сервисом которая предоставляет политику доступа по которой:

- Все могут просматривать профили
- Пользователи могут создавать/обновлять только свои профили

### **Функции:**

- Аутентификация через email и пароль.
- Восстановление пароля через ОТР.

# Описание дизайна приложения

## 1. Главного экрана

Главный экран приложения (рис. 3) на нем реализовано отображение данных пользователя переход на экран избранного и отображении списка отелей:



Рисунок 3 Главный экран приложения

## 2. Экран отображения бронирований.

Экран отображения бронирований изображён на (рис. 4) на этом экране отображаются апартаменты и итоговая цена за проживание на определенный период времени.



Рисунок 4 Экран мои бронирования

### 3. Экран поиска и фильтрации отелей

Экран поиска изображен на (рис. 5) на нем реализован поиск по названию отеля для быстрого нахождения нужного отеля, а также фильтрация по нескольким критериям таким как: количество звёзд, место положение , и по контексту описания.

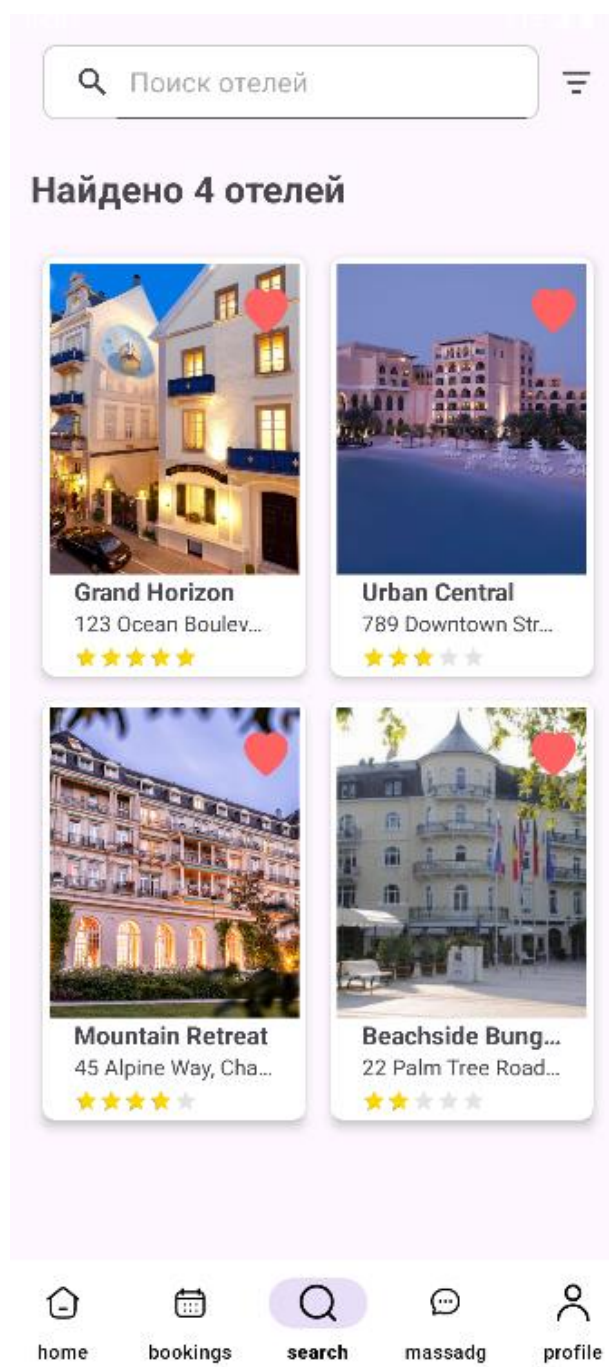


Рисунок 5 Экран поиска

#### **4. Экран для отображения чатов**

Экран чатов пользователя изображен на (рис. 6) На нем отображаются все чаты в которых участвует пользователь, чаты создаются после бронирования нового отеля с его менеджером для уточнения каких либо вопросов.



Рисунок 6 Экран чатов пользователя

## 5. Экран профиля пользователя

Экран профиля изображен на (рис.7) этот экран отображает данные пользователя позволяет выйти из аккаунта и редактировать данные.

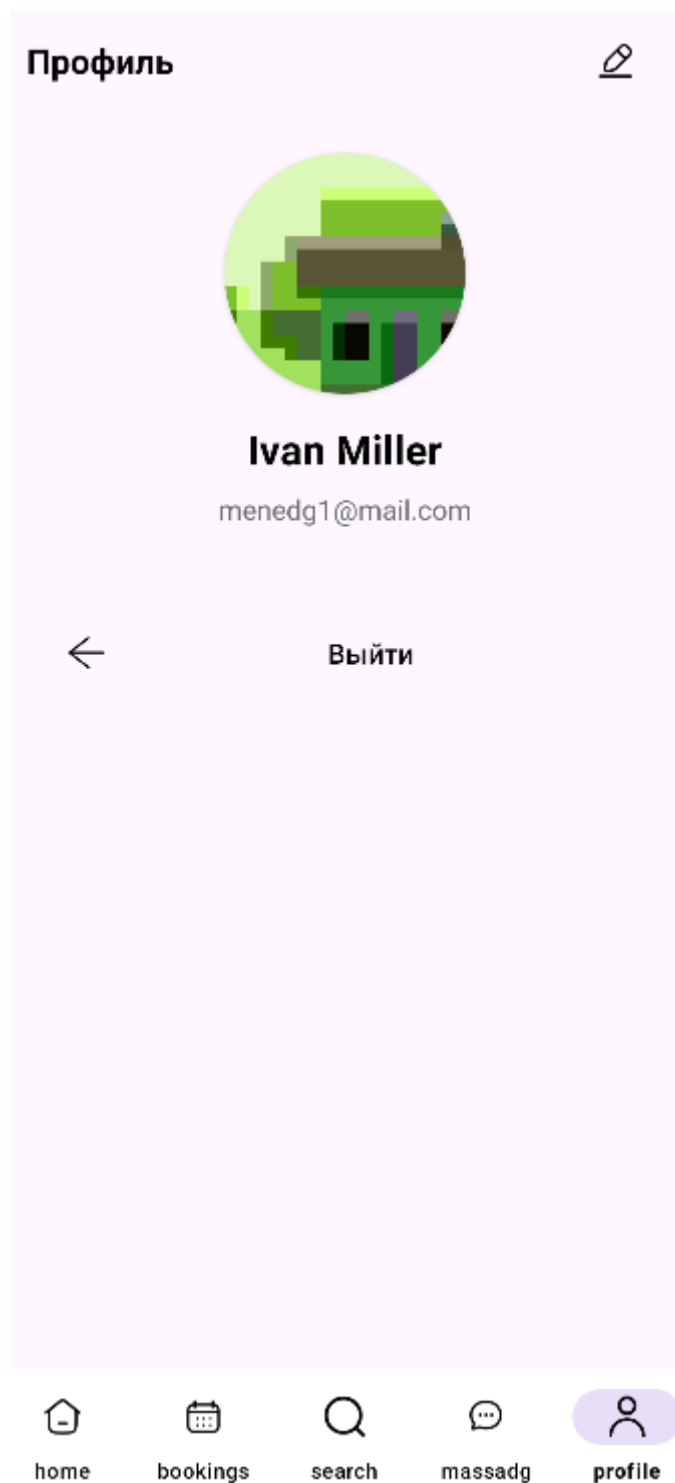
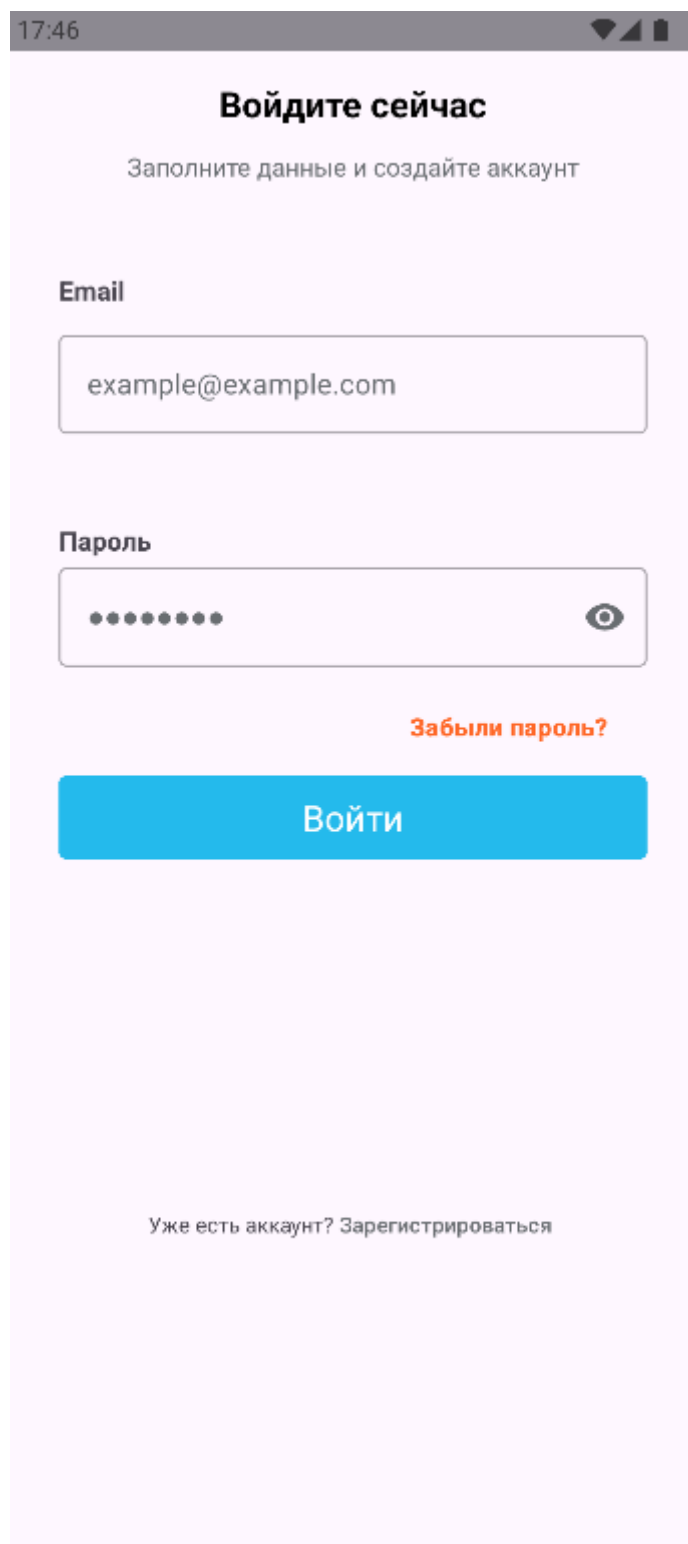


Рисунок 7 Экран профиля

## 6. Экран входа в приложение

На этом экране расположены элементы для ввода пароля и email, а также кнопки для перехода к экрану регистрации и восстановлению пароля через ОТП. (рис. 8)



17:46

## Войдите сейчас

Заполните данные и создайте аккаунт

Email

example@example.com

Пароль

••••••••

[Забыли пароль?](#)

Войти

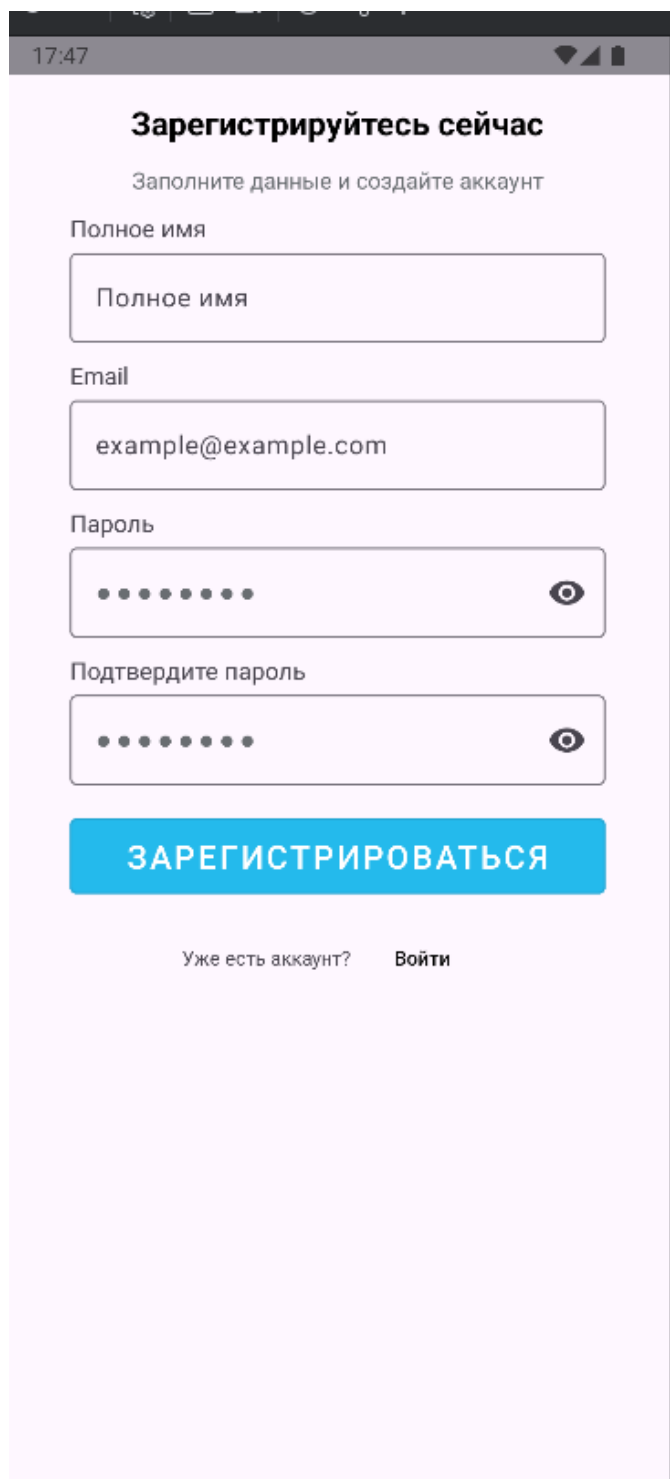
Уже есть аккаунт? [Зарегистрироваться](#)

Рисунок 8 Эcran входа

## 7. Эcran регистрации



На экране регистрации расположены элементы для ввода пароля и его подтверждения ввода email и имени пользователя а также кнопка для перехода к входу в аккаунт если он уже создан(рис. 9)



17:47


## Зарегистрируйтесь сейчас

Заполните данные и создайте аккаунт


Полное имя

Email

Пароль

Подтвердите пароль

**ЗАРЕГИСТРИРОВАТЬСЯ**

Уже есть аккаунт? **Войти**

Рисунок 9 Экран регистрации

## 8. Экран избранного

На экране расположен список в котором отображаются элементов который пользователь добавил в избранное по ним можно перейти и убрать из избранного(рис. 10)

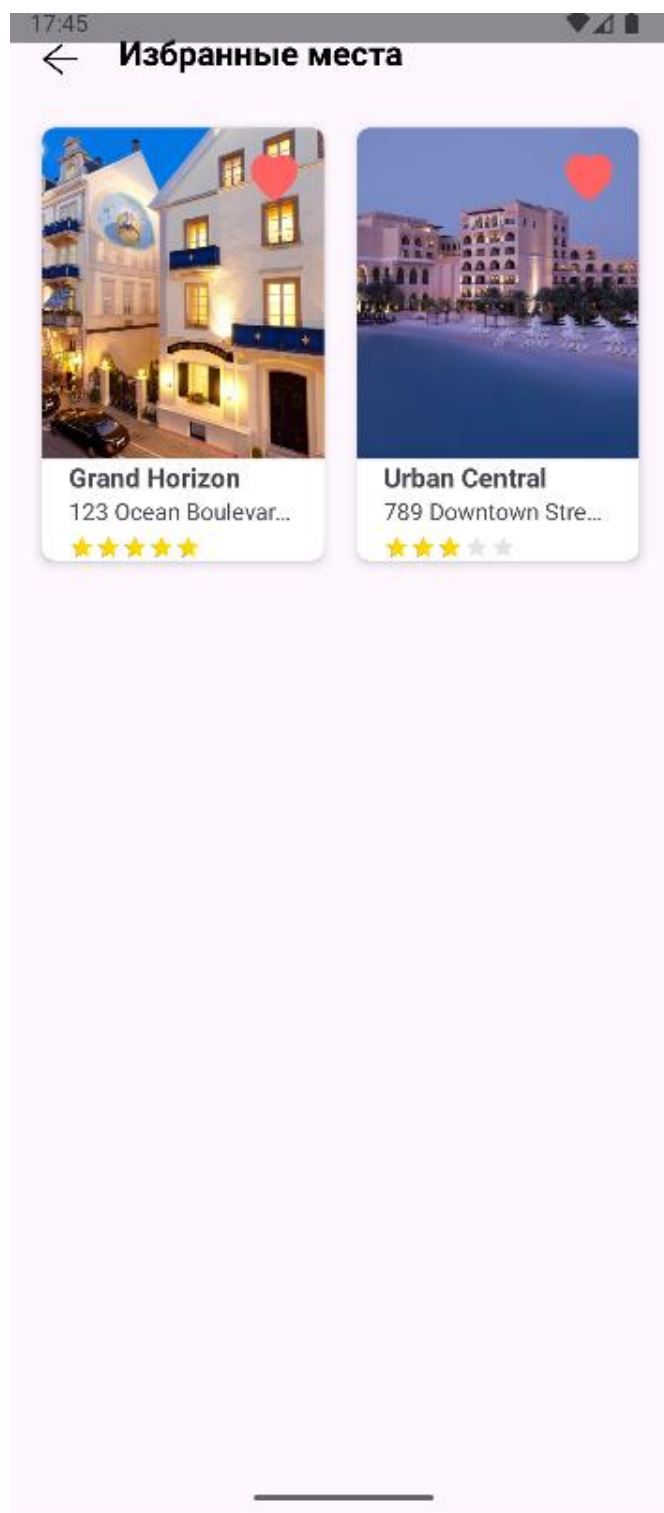
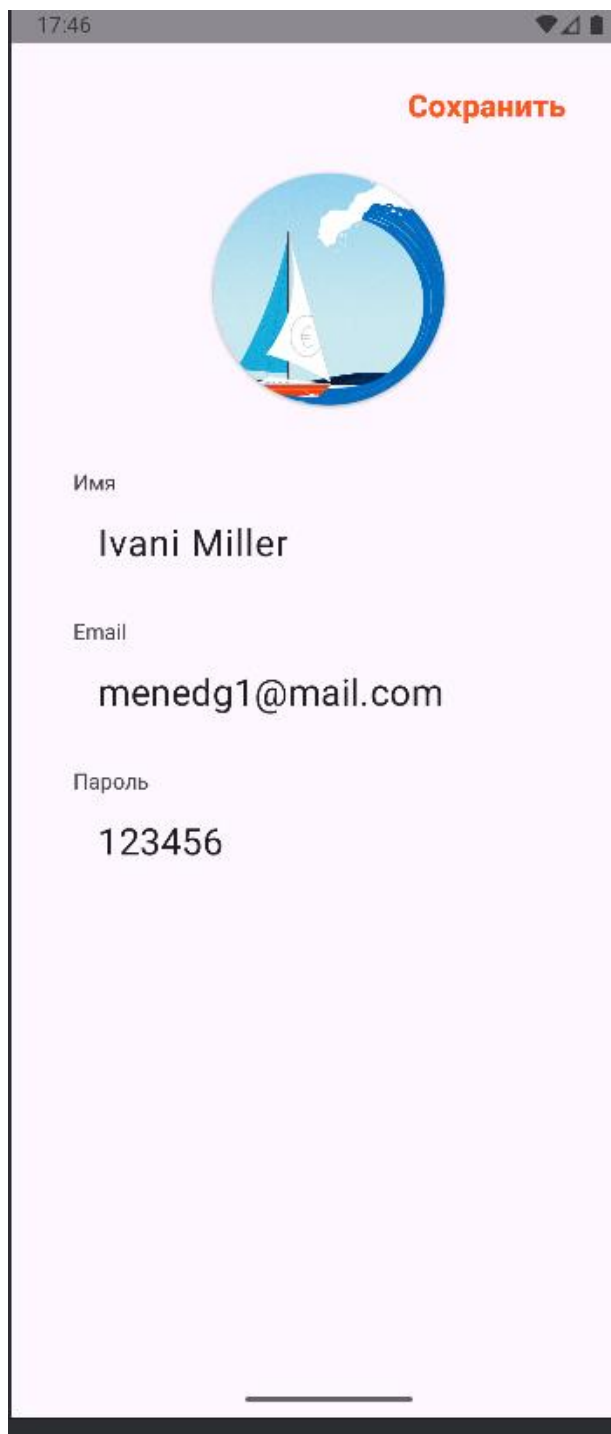


Рисунок 10 Экран избранного


## 9. экран редактирования профиля

На этом экране можно поменять свой профиль например: изменить фото, имя, пароль и почту (рис. 11)



17:46

Сохранить



Имя

Ivani Miller

Email

menedg1@mail.com

Пароль

123456

Рисунок 11 Экран изменения профиля

## Реализация функций приложения

### Функция добавления в избранное (addFavorite) листинг 1

#### Назначение:

Функция addFavorite предназначена для добавления отеля в список избранного пользователя. Она отправляет запрос на сервер, связывая идентификатор пользователя userId с идентификатором отеля hotelId в таблице избранного.

#### Логика работы:

1. Создается JSON-объект с параметрами:
  - user\_id – ID пользователя, добавляющего отель в избранное.
  - hotel\_id – ID отеля, который нужно добавить.
2. Отправляет POST-запрос на серверный на эндпоинт favorites

#### Литсинг 1:

```
void addFavorite(String userId, String hotelId, final SimpleCallback
callback) {
    String json = "{ \"user_id\": \"" + userId + "\", \"hotel_id\": \"" +
hotelId + "\" }";
    MediaType JSON = MediaType.get("application/json; charset=utf-8");

    RequestBody body = RequestBody.create(JSON, json);

    String url = DOMAIN_NAME + REST_PATH + "favorites";

    Request request = new Request.Builder()
        .url(url)
        .post(body)
        .addHeader("apikey", API_KEY)
```

```

        .addHeader("Authorization", DataBindingUtil.getBearerToken())
        .addHeader("Prefer", "return=minimal")
        .build();

client.newCall(request).enqueue(new Callback() {
    @Override
    public void onFailure(@NonNull Call call, @NonNull IOException e)
    {
        callback.onError(e);
    }

    @Override
    public void onResponse(@NonNull Call call, @NonNull Response
response) throws IOException {
        if (response.isSuccessful()) {
            callback.onSuccess();
        } else {
            callback.onError(new IOException("Не удалось добавить в
избранное"));
        }
    }
});
}

```

## Функция получения списка отелей (fetchHotels) листинг 2

### Назначение:

Функция fetchHotels выполняет запрос к серверу для получения полного списка отелей, доступных в системе. Она предназначена для отображения отелей в мобильном приложении.

### Логика работы:

#### 1. Формирование запроса:

- Создается GET-запрос к эндпоинту Hotels с параметром select=\* (получение всех полей)
- URL формируется как: {DOMAIN\_NAME}/rest/v1/Hotels?select=\*

## 2. Настройка заголовков:

- Добавляется обязательный заголовок apikey для доступа к API
- Включается токен авторизации через Authorization: Bearer {token}

## 3. Выполнение запроса:

- Запрос выполняется асинхронно через OkHttp
- Используется callback-интерфейс для обработки результата

### Листинг 2

```
public void fetchHotels(final SBC_Callback callback) {
    String url = DOMAIN_NAME + REST_PATH + "Hotels?select=*";

    Request request = new Request.Builder()
        .url(url)
        .get()
        .addHeader("apikey", API_KEY)
        .addHeader("Authorization", DataBinding.getBearerToken())
        .build();

    client.newCall(request).enqueue(new Callback() {
        @Override
        public void onFailure(@NonNull Call call, @NonNull IOException e)
        {
            callback.onFailure(e);
        }

        @Override
        public void onResponse(@NonNull Call call, @NonNull Response
response) throws IOException {
            if (response.isSuccessful() && response.body() != null) {
                String responseBody = response.body().string();
            }
        }
    });
}
```

```
        callback.onResponse(responseBody);
    } else {
        callback.onFailure(new IOException("Failed to fetch
hotels"));
    }
}
});
}
```

## Функция обновления профиля (updateProfile) листинг 3

### Назначение:

Функция updateProfile выполняет частичное обновление данных пользовательского профиля в базе данных через REST API. Она отправляет измененные данные на сервер и обрабатывает результат операции.

#### 1. Подготовка данных:

- Конвертирует объект ProfileUpdate в JSON-строку с помощью Gson
- Создает тело запроса (RequestBody) с типом содержимого application/json

#### 2. Формирование запроса:

- Использует PATCH-метод для частичного обновления
- URL включает фильтр по ID пользователя: profiles?id=eq.{userId}
- Добавляет обязательные заголовки:
  - apikey - ключ доступа к API
  - Authorization - Bearer Token для аутентификации

#### 3. Выполнение запроса:

- Запрос выполняется асинхронно через OkHttp
- Результат возвращается через callback-интерфейс

### Листинг 3

```
public void updateProfile(String userId, ProfileUpdate profileUpdate,
SBC_Callback callback) {

    MediaType JSON = MediaType.get("application/json; charset=utf-8");

    Gson gson = new Gson();

    String json = gson.toJson(profileUpdate);

    RequestBody body = RequestBody.create(JSON, json);

    Request request = new Request.Builder()

        .url(DOMAIN_NAME + REST_PATH + "profiles?id=req." + userId)

        .patch(body)

        .addHeader("apikey", API_KEY)

        .addHeader("Authorization", DataBinding.getBearerToken())

        .build();

    client.newCall(request).enqueue(new Callback() {

        @Override

        public void onFailure(@NonNull Call call, @NonNull IOException

e) {

            callback.onFailure(e);

        }

        @Override

        public void onResponse(@NonNull Call call, @NonNull Response

response) throws IOException {

            if (response.isSuccessful()) {
```



```
        callback.onResponse("OK");

    } else {

        callback.onFailure(new IOException("Ошибка сервера"));

    }

}

});

}
```

## Функция удаления сообщения (deleteMessage) листинг 4

### Назначение:

- Функция deleteMessage выполняет удаление сообщения из базы данных через REST API Supabase. Она отправляет DELETE-запрос на сервер и обрабатывает результат операции, обновляя UI в зависимости от ответа.

### Логика работы:

#### Подготовка данных:

- Извлекает messageId из переданного объекта Message.
- Формирует URL запроса с фильтром по ID сообщения: messages?id=eq.{messageId}.

#### Формирование запроса:

- Использует DELETE-метод для удаления записи.
- Добавляет обязательные заголовки для аутентификации в Supabase:
- apikey — ключ доступа к API.
- Authorization — Bearer Token.

#### Выполнение запроса:

- Запрос выполняется асинхронно через OkHttp.
- В случае успеха (response.isSuccessful()) сообщение удаляется из списка messageList, и UI обновляется.
- При ошибке выводится соответствующее уведомление.

#### Листинг 4

```
private void deleteMessage(Message message) {

    int messageId = message.getId();

    String url = "https://mmbdesfnabtcnpjwcwde.supabase.co/rest/v1/messages?id=eq." + messageId;

    Request request = new Request.Builder()

        .url(url)

        .delete()

        .addHeader("apikey",
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6Im1tYmRlc2ZuYWJ0Y2Jwandjd2Rlliwicm9sZSI6ImFub24iLCJpYXQiOiJlE3NDg5NTg4MDMsImV4cCI6ImJA2NDUzNDgwM30.zU9xsd7HmVuLi6OkiKTaB723ek2YNomMgrqnKKvSvQk")

        .addHeader("Authorization", "Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6Im1tYmRlc2ZuYWJ0Y2Jwandjd2Rlliwicm9sZSI6ImFub24iLCJpYXQiOiJlE3NDg5NTg4MDMsImV4cCI6ImJA2NDUzNDgwM30.zU9xsd7HmVuLi6OkiKTaB723ek2YNomMgrqnKKvSvQk")

        .build();

    client.newCall(request).enqueue(new Callback() {

        @Override

        public void onFailure(Call call, IOException e) {

            runOnUiThread(() -> Toast.makeText(ChatDetailActivity.this, "Ошибка при удалении",
Toast.LENGTH_SHORT).show());

        }

        @Override

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

```

        if (response.isSuccessful()) {

            runOnUiThread(() -> {

                int position = messageList.indexOf(message);

                if (position != -1) {

                    messageList.remove(position);

                    messageAdapter.notifyItemRemoved(position);

                    Toast.makeText(ChatDetailActivity.this, "Сообщение удалено",
Toast.LENGTH_SHORT).show();

                }

            });

        } else {

            runOnUiThread(() -> Toast.makeText(ChatDetailActivity.this, "Не удалось удалить",
Toast.LENGTH_SHORT).show());

        }

    }

});

}

```

## Функция обновления бронирования (updateExistingBooking) листинг 5

### Назначение:

- Функция updateExistingBooking выполняет обновление данных существующего бронирования через REST API Supabase. Она отправляет PATCH-запрос с новыми параметрами брони (даты, количество гостей, стоимость и др.) и обрабатывает результат операции.

### Логика работы:

## 1. Проверка входных данных

- Получает booking\_id из Intent (если отсутствует — завершает выполнение).
- Проверяет, что booking\_id не пустой.

## 2. Настройка HTTP-клиента (OkHttp)

- Добавляет необходимые заголовки для работы с Supabase API:

## 3. Выполнение запроса

- Отправка PATCH-запроса через SupabaseApiService.
- Фильтрация по ID бронирования (eq. + bookingId).

### Листинг 5

```
private void updateExistingBooking(double totalSum, String checkInDate, String checkOutDate,
                                int guestsCount, int roomId) {
    String bookingId = getIntent().getStringExtra("booking_id");
    if (bookingId == null || bookingId.isEmpty()) return;

    OkHttpClient client = new OkHttpClient.Builder()
        .addInterceptor(chain -> {
            okhttp3.Request original = chain.request();
            okhttp3.Request request = original.newBuilder()
                .header("apikey", API_KEY)
                .header("Authorization", "Bearer " + API_KEY)
                .header("Content-Type", "application/json")
                .header("Prefer", "return=representation")
                .method(original.method(), original.body())
                .build();
            return chain.proceed(request);
        })
        .build();
}
```

```

Retrofit retrofit = new Retrofit.Builder()

    .baseUrl(SUPABASE_URL)

    .client(client)

    .addConverterFactory(GsonConverterFactory.create())

    .build();

BookingRequest booking = new BookingRequest(

    DataBinding.getUuidUser(),

    checkInDate,

    checkOutDate,

    guestsCount,

    roomId,

    Integer.parseInt(hotelId)

);

booking.setSumm(totalSum);

SupabaseApiService apiService = retrofit.create(SupabaseApiService.class);

Call<Void> call = apiService.updateBooking("eq." + bookingId, booking);

call.enqueue(new Callback<Void>() {

    @Override

    public void onResponse(Call<Void> call, Response<Void> response) {

        if (response.isSuccessful()) {

            Toast.makeText(BookingFormActivity.this,

                "Бронирование успешно обновлено!", Toast.LENGTH_SHORT).show();

            finish();

        } else {

```

```

        Toast.makeText(BookingFormActivity.this,

            "Ошибка обновления бронирования", Toast.LENGTH_SHORT).show();

    }

}

@Override

public void onFailure(Call<Void> call, Throwable t) {

    Toast.makeText(BookingFormActivity.this,

        "Ошибка сети: " + t.getMessage(), Toast.LENGTH_SHORT).show();

    }

});

}

```

## Функция изменения email (changeEmail) листинг 6

### Назначение

- Обновляет email пользователя через вызов хранимой процедуры Supabase (change\_user\_email\_verified). Использует POST-запрос с JSON-телом.

### Логика работы

#### Подготовка данных:

- Формирует JSON с target\_user\_id (из сессии) и новым email.
- Создает RequestBody с типом application/json.

#### Заголовки:

- apikey и Authorization для доступа к API.
- Content-Type: application/json.

#### Обработка ответа:

- Успех: возвращает тело ответа через `callback.onResponse()`.
- Ошибка: передает исключение в `callback.onFailure()`.

## Листинг 6

```
public void changeEmail(Context context, String newEmail, SBC_Callback callback) {  
    JSONObject jsonBody = new JSONObject();  
    SessionManager sessionManager = new SessionManager(context);  
  
    try {  
        jsonBody.put("target_user_id", sessionManager.getUserId());  
        jsonBody.put("new_email", newEmail);  
    } catch (Exception e) {  
        e.printStackTrace();  
        return;  
    }  
  
    MediaType JSON = MediaType.get("application/json; charset=utf-8");  
    RequestBody body = RequestBody.create(JSON, jsonBody.toString());  
  
    Request request = new Request.Builder()  
        .url(DOMAIN_NAME + REST_PATH + "rpc/change_user_email_verified")  
        .post(body)  
        .addHeader("apikey", API_KEY)  
        .addHeader("Authorization", "Bearer " + sessionManager.getBearerToken())  
        .addHeader("Content-Type", "application/json")  
        .build();  
  
    client.newCall(request).enqueue(new Callback() {  
        @Override  
        public void onFailure(@NonNull Call call, @NonNull IOException e) {  
            callback.onFailure(e);  
        }  
    })
```

```

@Override

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

    if (response.isSuccessful()) {

        callback.onResponse(response.body().string());

    } else {

        callback.onFailure(new IOException("Ошибка сервера: " + response.code()));

    }

}

});

}

```

## Функция загрузки аватара (uploadAvatar) листинг 7

### Назначение

- Загружает изображение аватара в Supabase Storage (в папку avatars).

### Логика работы

#### Проверка файла:

- Конвертирует Uri в реальный путь через RealPathUtil.
- Если путь не найден — вызывает onFailure.

#### Подготовка запроса:

- Создает MultipartBody с файлом и именем.
- Использует PUT-запрос к /storage/v1/object/avatars/{fileName}.

#### Обработка ответа:

- Успех: возвращает ответ сервера.
- Ошибка: передает код статуса и тело ошибки.



## Листинг 7

```
public void uploadAvatar(Uri uri, String fileName, SBC_Callback callback, Context context) {
    String realPath = RealPathUtil.getRealPath(context, uri);
    if (realPath == null) {
        callback.onFailure(new IOException("Не удалось получить путь файла"));
        return;
    }

    File file = new File(realPath);

    RequestBody requestBody = RequestBody.create(MediaType.parse("image/*"), file);

    MultipartBody body = new MultipartBody.Builder()
        .setType(MultipartBody.FORM)
        .addFormDataPart("file", fileName, requestBody)
        .build();

    String url = DOMAIN_NAME + "/storage/v1/object/avatars/" + fileName;

    Request request = new Request.Builder()
        .url(url)
        .put(body)
        .addHeader("apikey", API_KEY)
        .addHeader("Authorization", DataBinding.getBearerToken())
        .build();

    client.newCall(request).enqueue(new Callback() {
        @Override
        public void onFailure(@NonNull Call call, @NonNull IOException e) {
            callback.onFailure(e);
        }

        @Override
        public void onResponse(@NonNull Call call, @NonNull Response response) throws IOException {
            if (response.isSuccessful()) {
                callback.onResponse(response.body().string());
            } else {
                String errorBody = response.body() != null ? response.body().string() : "Empty response";
                callback.onFailure(new IOException("Upload failed: " + response.code() + ", Body: " +
                    errorBody));
            }
        }
    });
}
```

## Функция получения списка отелей (fetchHotels) листинг 8

### Назначение

- Запрашивает список всех отелей из таблицы Hotels Supabase.

### Логика работы

#### Запрос:

- GET-запрос к Hotels?select=\*.
- Заголовки: apikey и Authorization.

#### Обработка ответа:

- Успех: возвращает JSON-список отелей.
- Ошибка: передает исключение.

### Листинг 8

```
public void fetchHotels(final SBC_Callback callback) {  
    String url = DOMAIN_NAME + REST_PATH + "Hotels?select=*";  
  
    Request request = new Request.Builder()  
        .url(url)  
        .get()  
        .addHeader("apikey", API_KEY)  
        .addHeader("Authorization", DataBinding.getBearerToken())  
        .build();  
  
    client.newCall(request).enqueue(new Callback() {  
        @Override  
        public void onFailure(@NonNull Call call, @NonNull IOException e) {  
            callback.onFailure(e);  
        }  
    })  
  
    @Override
```

```

public void onResponse(@NonNull Call call, @NonNull Response response) throws IOException {
    if (response.isSuccessful() && response.body() != null) {
        String responseBody = response.body().string();
        callback.onResponse(responseBody);
    } else {
        callback.onFailure(new IOException("Failed to fetch hotels"));
    }
}
});
}

```

## Функция получения деталей отеля (fetchHotelDetails) листинг 9

### Назначение

- Запрашивает детали конкретного отеля по его ID.

### Логика работы

#### Запрос:

- GET-запрос к Hotels?id=eq.{hotelId}.
- Заголовки: apikey и Authorization.

#### Обработка ответа:

- Успех: возвращает JSON с данными отеля.
- Ошибка: передает исключение.

## Листинг 9

```

public void fetchHotelDetails(String hotelId, final SBC_Callback callback) {
    String url = DOMAIN_NAME + REST_PATH + "Hotels?id=eq." + hotelId;
}

```

```

Request request = new Request.Builder()

    .url(url)

    .get()

    .addHeader("apikey", API_KEY)

    .addHeader("Authorization", DataBinding.getBearerToken())

    .build();

client.newCall(request).enqueue(new Callback() {

    @Override

    public void onFailure(@NonNull Call call, @NonNull IOException e) {

        callback.onFailure(e);

    }

    @Override

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

        if (response.isSuccessful() && response.body() != null) {

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

            callback.onResponse(responseBody);

        } else {

            callback.onFailure(new IOException("Failed to fetch hotel details"));

        }

    }

});
}

```

## **Заключение**

В рамках данной работы было разработано мобильное приложение "Trevanor", интегрирующее следующие функции бронирование отелей, общение с менеджером отелей и добавление в избранное. Проект успешно реализован в соответствии с поставленными задачами и современными стандартами мобильной разработки.

На начальном этапе было разработано детализированное техническое задание, включающее оптимизированную структуру данных и продуманный дизайн-макет в Figma, который стал основой для создания адаптивного интерфейса.

Серверная часть приложения построена на базе Supabase. Интерфейс приложения выполнен в единой стилистике. Особое внимание было уделено адаптивности – приложение корректно отображается на устройствах с различными размерами экранов.

Данная практика помогла нам закрепить приобретённые навыки в ходе обучения и узнать новые способы реализации решений.

## Список литературы

1. Харди, Б. Разработка приложений для Android. Полное руководство для начинающих. – 2-е изд. – СПб.: Питер, 2022.
2. Дейтел, П. Android 13 для разработчиков на Java. – М.: Эксмо, 2023.
3. Берк, К. Android-программирование для профессионалов. – 4-е изд. – М.: ДМК Пресс, 2023.
4. Гриффитс, Д. Программирование под Android. – 3-е изд. – СПб.: БХВ-Петербург, 2022.
5. Колисниченко, Д. Программирование для Android. – М.: БХВ-Петербург, 2011.
6. Коматинени, С. Android 3 для профессионалов. – М.: Вильямс, 2012.
7. Голощапов, А. Google Android. Программирование для мобильных устройств. – М.: БХВ-Петербург, 2012.
8. Хорстманн, К. Java. Библиотека профессионала. Том 2. – 12-е изд. – М.: Диалектика, 2023.
9. Блох, Дж. Эффективное программирование на Java для Android. – 3-е изд. – М.: ДМК Пресс, 2022.
10. Маклин, Г. Android Security: защита приложений на Java. – М.: Эксмо, 2023.