09.02.07 ПР-312

**ОТЧЕТ ПО ПРОИЗВОДСВЕННОЙ ПРАКТИКЕ**

**ПП.01.01 Разработка мобильных приложений**

**ПМ.01 Разработка модулей программного обеспечения**

|  |  |  |
| --- | --- | --- |
| **Практикант** |  | **Земцов М.А.** |
| **Руководители практики** |  | **Домбровский Н.С.**  **Смирнова Е.Е.** |
|  |  |  |

**Оглавление**

[**Задание на разработку 3**](#_Toc201281758)

[**Анализ индивидуального задания 4**](#_Toc201281759)

[**Проектирование дизайна приложения 5**](#_Toc201281760)

[**Описание дизайна приложения 9**](#_Toc201281761)

[**Реализация функций приложения 18**](#_Toc201281762)

[**Заключение 30**](#_Toc201281763)

[**Список литературы 32**](#_Toc201281764)

# Задание на разработку

Цель практики

# Разработка финансового мобильного приложения «TransferMe» для удобного управления переводами средств, контроля личных финансов и безопасной аутентификации пользователе*й.*

Структура задания

1. 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.

# Анализ индивидуального задания

Разработано мобильное приложение на Java с модульной архитектурой для взаимодействия пользователей с финансами и переводами.

**Модуль 1. Техническое задание**

Формулировка целей и задач, дизайн-макет, проектирование структуры данных в Supabase.

**Модуль 2. Серверная часть**

Используется Supabase для хранения данных, аутентификации и управления контентом. Реализованы механизмы восстановления пароля через OTP и политики безопасности, шифрование конфиденциальных данных.

**Модуль 3. Верстка интерфейса**

Реализованы экраны по макетам, адаптация под разные размеры экранов, навигация между экранами, обработка ошибок.

**Модуль 4. Функционал**

Реализована регистрация, авторизация (email + пароль), вход в приложение через PIN, взаимодействие с категориями, добавлением карт, перевод денежных средств, удаление истории, профиль пользователя, фильтрация данных, поддержка двух языков.

**Модуль 5. Тестирование**

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

**Модуль 6. Документация**

Создание отчёта, структурирование исходного кода.

# Проектирование дизайна приложения

Основные принципы:

1. Простота и ясность:

* Минимум текста, только ключевая информация
* Четкие заголовки и понятные иллюстрации

1. Адаптивность — интерфейс корректно отображается на разных размерах экранов, все элементы масштабируются без потери функциональности;
2. Доступность — соблюдены требования к контрастности, интерактивные элементы достаточно крупные для удобного использования.

**Изображение выглядит как снимок экрана, текст, программное обеспечение, Значок на компьютере

Содержимое, созданное искусственным интеллектом, может быть неверным.**

Рисунок 1 Макет Figma

Основные экраны интерфейса:

**1.1. Главный экран**

* Отображает список транзакций, карточек, а также баланса;
* В нижней панели доступен быстрый переход между основными страницами.

**1.2. Экран авторизации**

* Дизайн с формой входа, без лишних элементов;
* Поля для ввода почты и пароля;
* Кнопки дополнительных действий:
  + Восстановление пароля;
  + Переход к регистрации.

**1.3. Процесс регистрации**

* На форме расположены поля для ввода персональных данных (email, пароль, подтверждение пароля);
* Кнопка для перехода к авторизации.
* Если уже зарегистрированы и вас перенесло на этот экран, можно спокойно перейти на авторизацию

**1.4. Профиль пользователя**

* На странице профиля отображается:
  + Аватар пользователя;
  + Номер аккаунта
  + Никнейм, email, номер телефона, адрес;
  + Поля для изменения данных;
  + Быстрые действия (смена пароля);
  + Просмотр истории;
  + Поддержка;
  + Имеется кнопка выхода из аккаунта

**1.5. Взаимодействие с категориями**

* Список всех доступных категорий;
* Возможность перевода денег на данную категорию;

**1.6. Взаимодействие с картами**

* Добавление карточки лично;
* Возможен выбор цвета карточки;

**1.7. Сортировка истории**

* Позволяет сортировать по категориям;
* По одной либо нескольким сразу.

**1.8. Удаление истории**

* Кнопка по нажатии которой будет выполнено удаление всех историй транзакций.

# Описание данных приложения

Структура данных приложения разработана для обеспечения работы основных модулей:

* Управление данными пользователей;
* Учёт и обработка данных, транзакций по пользователю;
* Хранение информации о профилях пользователей и их истории;
* Хранение данных о транзакции по категориям;
* Хранение данных о банковских картах.

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

Изображение выглядит как снимок экрана, текст, дизайн

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 2 Схема БД

# Описание дизайна приложения

**Регистрация**

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 3 экран Регистрации

Экран регистрации нового пользователя:

* Поля для ввода email, пароля и подтверждение пароля;
* Кнопка "Регистрация" для создания аккаунта;
* Ссылка для перехода к экрану входа («Есть аккаунт?»), если аккаунт уже создан;
* Валидация всех полей, информирование пользователя об ошибках ввода.

**Экран входа**

Изображение выглядит как текст, снимок экрана, дизайн

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4 экран Входа

Экран входа в приложение содержит:

* Поля для ввода email и пароля;
* Кнопку «Авторизация» для авторизации, и перехода на страницу создания PIN-кода;
* Ссылку «Забыли пароль?» для перехода к восстановлению пароля;
* Ссылку для перехода на экран регистрации («Нет аккаунта?»).

**Главный экран**

Изображение выглядит как текст, снимок экрана, Реклама в Интернете, Веб-сайт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 5 Главный экран

Главный экран приложения со списками транзакций людей и карточками, снизу экрана расположено меню с переходами на другие экраны.

**Экран «Мой кошелек»**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 6 Экран категорий, карточек

На данном экране расположены:

* Категории, при нажатии на которые вы переходите на экран с возможным переводом денежных средств на ту категорию, которую выбрали;
* Нажав на + вы переходите на экран создания банковской карты;
* Список доступных карт можно пролистать и посмотреть.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 7 экран перехода на перевод по категории

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 8 Экран перехода на создание карты

**Экран Курса транзакций (в разработке)**

Изображение выглядит как текст, снимок экрана, дизайн

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 9 Экран *курса транзакций*

**Экран Профиля**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 10 экран настоек профиля

На экране профиля расположены элементы:

* Аватар, пользователь может добавить свою фотку в приложении;
* Можно поменять данные, а именно адрес, и никнейм;
* Можно поменять пароль;
* Присутствуют две кнопки Истории и Поддержки с которыми можно взаимодействовать
* А также есть кнопка выхода из аккаунта

**Экран Истории транзакций**

Изображение выглядит как текст, снимок экрана, дизайн

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 11 экран истории

На данном экране находятся категории на которые были переведены средства

* Доступен фильтр истории, и кнопка удаления;

Изображение выглядит как текст, снимок экрана, программное обеспечение, дизайн

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 12 Экран фильтрации

На экране корзины расположены элементы:

* В фильре можно выбрать категории по которым нужно отсортировать список истории

**Экран оформления заказа**

Изображение выглядит как текст, снимок экрана, Операционная система, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 13 Экран добавления карточки

На экране оформления карты расположены:

* Данные о карте;
* Есть возможность выбрать цвет карты.

# 

# Реализация функций приложения

**Входа пользователя (login)**

**Назначение:**  
Осуществляет аутентификацию пользователя по email и паролю, обеспечивает переход к экрану ввода и создания пин-кода либо главному экрану приложения.

**Логика работы:**

1. Введенные данные пользователем, отправляет в базу данных для проверки существования пользователя;
2. Позже сохраняет id и токен для дальнейшего использования;
3. Также сохраняет сессию и тип пользователя;
4. При успешной авторизации перекидывает на создание пин-кода.

**Листинг:**

public void login(LoginReguest loginRequest, final SBC\_Callback callback){  
  
 MediaType mediaType = MediaType.*parse*("application/json");  
 Gson gson = new Gson();  
 String json = gson.toJson(loginRequest);  
 RequestBody body = RequestBody.*create*(json, mediaType);  
 Request request = new Request.Builder()  
 .url(*DOMAIN\_NAME* + *AUTH\_PATH* + "token?grant\_type=password")  
 .method("POST", body)  
 .addHeader("apikey", *API\_KEY*)  
 .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()){  
 String responseBody = response.body().string();  
 callback.onResponse(responseBody);  
 }else {  
 callback.onFailure(new IOException("Ошибка сервера: " + response));  
 }  
 }  
 });  
  
}

**Регистрации нового пользователя (register)**

**Назначение:**  
Создаёт новый аккаунт пользователя с введенными данными и переходом к созданию пин-кода.

**Логика работы:**

1. Введенные данные пользователем (email, пароль) отправляет в базу данных для создания нового пользователя;
2. Сохраняет id во внутренних настройках;
3. Сохраняет тип данных;
4. Если регистрация успешная, отправляет на экран создания пин-кода;
5. В случае ошибки отображает сообщение.

**Листинг:**

public void register(LoginReguest loginRequest, final SBC\_Callback callback){  
 MediaType mediaType = MediaType.*parse*("application/json");  
 Gson gson = new Gson();  
 String json = gson.toJson(loginRequest);  
 RequestBody body = RequestBody.*create*(json, mediaType);  
 Request request = new Request.Builder()  
 .url(*DOMAIN\_NAME* + *AUTH\_PATH* + "signup")  
 .method("POST", body)  
 .addHeader("apikey", *API\_KEY* )  
 .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()){  
 String responseBody = response.body().string();  
 callback.onResponse(responseBody);  
 }else {  
 callback.onFailure(new IOException("Ошибка сервера: " + response));  
 }  
 }  
 });  
  
};

**Восстановление пароля (sendRecoveryRequest)**

**Назначение:**  
Позволяет пользователю сбросить забытый пароль через email с подтверждением по OTP.

**Логика работы:**

1. Пользователь вводит email для восстановления;
2. Email отправляется в базу данных для проверки и отправки кода;
3. После успешной отправки переходит на экран ввода кода OTP;
4. После ввода кода открывается экран ввода нового пароля;
5. После ввода нового пароля переход к экрану входа.

**Листинг:**

public void sendRecoveryRequest(InputEmail inputEmail, final SBC\_Callback callback) {  
 MediaType mediaType = MediaType.*parse*("application/json");  
 Gson gson = new Gson();  
 String json = gson.toJson(inputEmail);  
 RequestBody body = RequestBody.*create*(json, mediaType);  
 Request request = new Request.Builder()  
 .url(*DOMAIN\_NAME* + *AUTH\_PATH* + "otp")  
 .post(body)  
 .addHeader("apikey", *API\_KEY*)  
 .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()){  
 String responseBody = response.body().string();  
 callback.onResponse(responseBody);  
 }else {  
 callback.onFailure(new IOException("Ошибка сервера: " + response));  
 }  
 }  
 });  
}

**Подтверждения OTP-кода (verifyCode)**

**Назначение:**  
Пользователь подтверждает восстановление пароля с помощью одноразового кода, отправленного на почту.

**Логика работы:**

1. Пользователь получает код OTP на свою почту;
2. После ввода кода и нажатия "Далее", код отправляется в базу данных для проверки;
3. При успешной проверке выполняется переход на экран смены пароля;
4. В случае ошибки отображается уведомление.

**Листинг:**

public void verifyOtp(String email, String code, final SBC\_Callback callback) {  
  
 MediaType mediaType = MediaType.*parse*("application/json");  
 String json = String.*format*(  
 "{\"type\":\"email\",\"email\":\"%s\",\"token\":\"%s\"}",  
 email, code  
 );  
 RequestBody body = RequestBody.*create*(json, mediaType);  
 Request request = new Request.Builder()  
 .url(*DOMAIN\_NAME* + *AUTH\_PATH* + "verify")  
 .post(body)  
 .addHeader("apikey", *API\_KEY\_INSTANCE\_DELETE*)  
 .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()){  
 String responseBody = response.body().string();  
 callback.onResponse(responseBody);  
 }else {  
 callback.onFailure(new IOException("Ошибка сервера: " + response));  
 }  
 }  
 });  
}

**Управление картами (getCards, addCards)**

**Назначение:**  
Позволяет просматривать, добавлять карты

**Логика работы:**

1. Загрузка списка банковских карт;
2. Возможность добавления карт в список;

**Листинг:**

public void getCards(String userId, final SBC\_Callback callback) {  
 Request request = new Request.Builder()  
 .url(*DOMAIN\_NAME* + *REST\_PATH* + "cards?id\_user=eq." + userId)  
 .addHeader("apikey", *API\_KEY*)  
 .addHeader("Authorization", "Bearer " + *API\_KEY*)  
 .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()) {  
 String responseBody = response.body().string();  
 callback.onResponse(responseBody);  
 } else {  
 callback.onFailure(new IOException("Ошибка сервера: " + response.code()));  
 }  
 }  
 });  
}

public void addCards(AddCard addCard, final SBC\_Callback callback){  
 MediaType mediaType = MediaType.*parse*("application/json");  
 Gson gson = new Gson();  
 String json = gson.toJson(addCard);  
 RequestBody body = RequestBody.*create*(json, mediaType);  
 Request request = new Request.Builder()  
 .url(*DOMAIN\_NAME* + *REST\_PATH* + "cards")  
 .post(body)  
 .addHeader("apikey", *API\_KEY*)  
 .addHeader("Authorization", "Bearer " + *API\_KEY*)  
 .addHeader("Content-Type", "application/json")  
 .addHeader("Prefer", "return=minimal")  
 .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()){  
 String responseBody = response.body().string();  
 callback.onResponse(responseBody);  
 }else {  
 callback.onFailure(new IOException("Ошибка сервера: " + response));  
 }  
 }  
 });  
}

**Управление категориями (getTransactions, addTransaction, fetchAllCategoryFilters)**

**Назначение:**  
Осуществляет перевод на категорию и вывод данных о данной транзакции, просмотр истории с фильтром

**Логика работы:**

1. Пользователь выбирает нужную категорию, нажимает иконку;
2. Происходит переход к экрану транзакции;
3. Осуществляем перевод по кнопке;
4. Заходим в историю и по кнопке фильтр выпадает диалоговое окно, в котором мы можем выбрать то что нам надо

**Листинг:**

public void getTransactions(SBC\_Callback callback) {  
 Request request = new Request.Builder()  
 .url(*DOMAIN\_NAME* + *REST\_PATH* + "history\_tranz?select=\*,Category(\*)")  
 .addHeader("apikey", *API\_KEY*)  
 .addHeader("Authorization", "Bearer " + *API\_KEY*)  
 .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()));  
 }  
 }  
 });  
}  
public void addTransaction(TranzAdd tranzAdd, final SBC\_Callback callback){  
 MediaType mediaType = MediaType.*parse*("application/json");  
 Gson gson = new Gson();  
 String json = gson.toJson(tranzAdd);  
 RequestBody body = RequestBody.*create*(json, mediaType);  
 Request request = new Request.Builder()  
 .url(*DOMAIN\_NAME* + *REST\_PATH* + "history\_tranz")  
 .post(body)  
 .addHeader("apikey", *API\_KEY*)  
 .addHeader("Authorization", "Bearer " + *API\_KEY*)  
 .addHeader("Content-Type", "application/json")  
 .addHeader("Prefer", "return=minimal")  
 .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()){  
 String responseBody = response.body().string();  
 callback.onResponse(responseBody);  
 }else {  
 callback.onFailure(new IOException("Ошибка сервера: " + response));  
 }  
 }  
 });  
}

public void fetchAllCategoryFilters(String query, final SBC\_Callback callback){  
 String queryFull = (query != null && !query.isEmpty() ? "&" + query : "");  
 Request request = new Request.Builder()  
 .url(*DOMAIN\_NAME* + *REST\_PATH* + "history\_tranz?select=\*,Category(\*)" + queryFull)  
 .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()){  
 String responseBody = response.body().string();  
 callback.onResponse(responseBody); }else {  
 callback.onFailure(new IOException("Ошибка сервера: " + response)); }  
 } });  
 }

**Удаление данных** **(deleteAllHistory)**

**Назначение:**  
Позволяет пользователю удалить данные.

**Логика работы:**

1. Пользователь нажимает на кнопку удалить;
2. И удаляются всё категорий которые были в истории.

**Листинг:**

public void deleteAllHistory(final SBC\_Callback callback){  
 Request request = new Request.Builder()  
 .url(*DOMAIN\_NAME* + *REST\_PATH* + "history\_tranz?id\_profiles=eq." + DataBinding.*getUuidUser*())  
 .delete()  
 .addHeader("apikey", *API\_KEY*)  
 .addHeader("Authorization", "Bearer " + *API\_KEY*)  
 .addHeader("Prefer", "return=minimal")  
 .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()){  
 String responseBody = response.body().string();  
 callback.onResponse(responseBody);  
 }else {  
 callback.onFailure(new IOException("Ошибка сервера: " + response));  
 }  
 }  
 });  
}

**Управление профилем пользователя обновление пароля (updateProfile)**

**Назначение:**  
Позволяет сделать восстановления пароля.

**Логика работы:**

1. Загрузка данных пользователя;
2. Возможность редактирования данных (никнейма, адреса);
3. Если будет восстановление пароля, то это происходит через OTP код;

**Листинг:**

public void updatePassword(String token, String email, String newPassword, final SBC\_Callback callback) {  
 MediaType mediaType = MediaType.*parse*("application/json");  
 String json = String.*format*(  
 "{\"email\":\"%s\",\"password\":\"%s\"}",  
 email, newPassword  
 );  
 RequestBody body = RequestBody.*create*(json, mediaType);  
 Request request = new Request.Builder()  
 .url(*DOMAIN\_NAME* + *AUTH\_PATH* + "user")  
 .put(body)  
 .addHeader("apikey", *API\_KEY*)  
 .addHeader("Authorization", "Bearer " + token)  
 .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()){  
 String responseBody = response.body().string();  
 callback.onResponse(responseBody);  
 }else {  
 callback.onFailure(new IOException("Ошибка сервера: " + response));  
 }  
 }  
 });  
}

# Заключение

В ходе практики мной было разработано мобильное приложение «TransferMe» на Java с использованием модульной архитектуры, которое включает следующие ключевые функции:

Система аутентификации:

* + Полноценный цикл регистрации и входа пользователей;
  + Восстановление доступа через email;
  + Защита учетных записей с помощью пин-кода.

Управление банковскими картами, а также оплатой категорий:

* + Добавление и настройка карт
* Оплата категорий

Управление историей транзакций:

* Выбор нужной категории используя фильтр;
* Просмотр информации сделанной за всё время
* Также возможность удаления данных

Персонализация:

* + Настройка профиля пользователя;
  + Смена данных;
  + Восстановление пароля.

Интерфейс и производительность:

* + Современный дизайн;
  + Интуитивно понятная навигация.

Работа над приложением мне помогла освоить работу с API, я научился передавать и принимать данные, было очень тяжело, но я справился и сделал всё возможное в моих силах и считаю у меня получилось хорошее приложение.

# Список литературы

# Jetpack Compose: Основы // Официальная документация Android Developers. URL: https://developer.android.com/jetpack/compose (дата обращения: 20.05.2025).

# Supabase для Android // Официальная документация Supabase. URL: https://supabase.com/docs/guides/android (дата обращения: 18.05.2025).

# Material Design 3 // Официальные гайдлайны Google. URL: https://m3.material.io (дата обращения: 22.05.2025).

# Архитектура Android-приложений // Android Developers. URL: https://developer.android.com/topic/architecture (дата обращения: 15.05.2025).

# Разработка под Android для начинающих // Habr. URL: https://habr.com/ru/articles/android-basics (дата обращения: 10.05.2025).

# Современные тренды в Android-разработке // ProAndroidDev. URL: https://habr.com/ru/articles/502282/ (дата обращения: 03.05.2025).

# UI/UX для мобильных приложений // Nielsen Norman Group. URL: https://www.nngroup.com/articles/mobile-ux (дата обращения: 05.05.2025).