МИША

Добрый день уважаемая комиссия!

Наша команда состоит из Саргина Михаила Александровича и Рябчиковой Яны Павловны, факультет общего и профессионального образования, наш проект — мобильное приложение AnimeBox.

Это результат нескольких месяцев изучения Android разработки, обсуждений, проб и ошибок, и, честно говоря, мы гордимся тем, что получилось.

ЯНА

Объединив усилия мы старались делать максимально хорошо исходя из полученных на курсе знаний.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0. Как мы работали в команде

МИША

У нас команда из двух человек, и мы сразу договорились о разделении ролей, но при этом активно помогали друг другу. Я Саргин Михаил Александрович в основном отвечал за техническую часть: писал большую часть кода, разбирался с архитектурой, интегрировал библиотеки, настраивал навигацию, работал с базой данных и API-интеграцией.

ЯНА

Я, Рябчикова Яна ПавловнаВторой участник больше занимался дизайном и UI/UX: создавал макеты экранов, подбирал цветовые схемы, работал с Material Design 3, а также выступал как project manager — планировал задачи, следил за сроками, оформлял документацию, делал презентацию и отчет по проекту.

МИША

Мы вместе писали код — когда я застревал на сложных моментах (например, с DI или Room), Яна помогала искать решения, вместе читали документацию, смотрели видеоуроки. А когда нужно было реализовать сложные UI-компоненты или анимации, я помогал с логикой, а второй участник — с визуальной частью. Например, анимацию получения персонажа мы делали вместе: я писал логику анимации, а второй участник подбирал цвета и эффекты.

ЯНА

Такое разделение помогло нам не только быстрее двигаться, но и не терять фокус: я мог глубже погрузиться в технические детали, а второй участник — в дизайн и пользовательский опыт. Благодаря этому мы смогли создать не только функциональное, но и визуально привлекательное приложение, которое понравится фанатам аниме. В итоге мы оба научились не только программировать и быстро изучать новые и сложные для нас темы.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

МИША

1. Введение и мотивация

Выбор тем был свободный, можно было выбрать из представленных тем преподавателем. Идея нашего приложения собственная — она возникла не случайно. Аниме в последние годы все больше набирает популярность, наш одногруппник-один из любителей японской культуры очень любит коллекционирование и исходя из наших исследований, эти группы довольно часто пересекаются, поэтому мы и решили создать приложение для них.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ЯНА

2. Основной функционал и пользовательский сценарий

Мы старались, чтобы каждый экран приложения решал конкретную задачу пользователя. Реализованы экраны:

● Главный экран — лутбоксы с персонажами из разных аниме, система кристаллов для открытия коробок.

● Экран коллекции — просмотр собранных персонажей с фильтрацией по редкости.

● Поиск аниме — интеграция с Jikan API для поиска информации об аниме.

● Экран деталей аниме — подробная информация, рейтинг, описание.

● Настройки — переключение темы, очистка коллекции.

● Диалоги — анимация получения персонажа, просмотр рекламы для кристаллов.

Каждый экран мы обсуждали с другими фанатами аниме и дорабатывали по их отзывам.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

МИША

3. Демонстрация экранов

(Показываем презентацию или приложение в эмуляторе)

● Главный экран — лутбоксы с персонажами из 7 популярных аниме, отображение баланса кристаллов.

● Открытие лутбокса — анимация получения персонажа с различной редкостью.

● Коллекция — сетка персонажей с информацией о редкости.

● Поиск аниме — поиск через Jikan API, отображение результатов.

● Настройки — переключение между светлой и темной темами.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\

ЯНА

4. Архитектура и структура проекта

Мы выбрали архитектуру MVVM + Clean Architecture — честно говоря изначально мы думали, что можно реализовать через MVC/MVI подходы, но по мере получения знаний стало понятно, что приложение такого уровня требует более сложной архитектуры.

В проекте:

● Presentation Layer — все экраны на Jetpack Compose, каждый экран — отдельный Composable, логика работы экранов вынесена во ViewModel.

● Domain Layer — use cases, которые инкапсулируют бизнес-логику. Например, открытие лутбокса — отдельный use case, который проверяет баланс кристаллов и генерирует персонажа.

● Data Layer — репозитории, которые общаются с Room и Jikan API.

ПРИМЕР КОДА МОЖЕШЬ НЕ ЧИТАТЬ

Пример кода:

```kotlin

class AnimeRepositoryImpl @Inject constructor(

private val local: CharacterDao,

private val remote: AnimeApi

) : AnimeRepository {

override suspend fun getAnimeList(query: String): List<Anime> {

return try {

val remoteData = remote.searchAnime(query)

remoteData

} catch (e: Exception) {

emptyList()

}

}

}

```

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

МИША

5. Реализация: детали и этапы

● Весь UI построен на Jetpack Compose с Material Design 3. Например, экран коллекции — это LazyVerticalGrid с карточками персонажей.

● Навигация реализована через Navigation Compose, все маршруты вынесены в отдельный файл.

● Для хранения данных о персонажах и кристаллах — отдельные сущности и DAO в Room.

● Для поиска аниме — интеграция с Jikan API через Retrofit.

● Для загрузки изображений персонажей — Coil.

● Для внутриигровой экономики — система кристаллов с получением за просмотр рекламы.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Почему именно такой стек

Все выбранные технологии мы брали из уроков курса. Мы считаем этот стек наиболее подходящим, потому что он современный, поддерживается Google, и реально работает в боевых проектах. К тому же, все модули курса — архитектура, базы данных, асинхронность, DI, UI — мы применили на практике.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7. Где и как использовали технологии

ЯНА

● Room — для локального хранения персонажей и кристаллов. При отсутствии интернета пользователь все равно видит свою коллекцию.

● DI (Hilt) — для внедрения зависимостей между слоями.

● Jikan API — для поиска информации об аниме.

● Coroutines/Flow — для асинхронной загрузки данных и обновления UI.

● Navigation Compose — для переходов между экранами.

● Coil — для загрузки изображений персонажей.

● Material Design 3 — для современного UI и поддержки темной темы.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8. Трудности и решения

МИША

Честно скажу, не все темы дались легко. Особенно сложно было с:

● Room — сначала путались с миграциями и связями между сущностями.

● DI (Hilt) — не сразу поняли, как правильно настраивать модули.

● API интеграция — пришлось разбираться с Retrofit и обработкой ошибок.

● Геймификация — было сложно реализовать систему редкости персонажей и анимации.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

9. Перспективы развития

В будущем хотим добавить:

● Систему достижений и ежедневных наград

● Звуковые эффекты при открытии коробок

● Больше аниме и персонажей

● Социальные функции (друзья, рейтинги)

● Мультиплатформенность (iOS, Web)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10. Личный опыт и выводы

ЯНА

В команде работало 2 человека: один отвечал за анализ и UX, второй — за разработку, хотя мы и часто помогали друг другу и по сути менялись ролями. За время работы мы научились не только программировать, но и работать в команде, планировать задачи, искать компромиссы.

МИША

Я лично понял, как важно продумывать архитектуру заранее, как интересно реализовывать геймификацию и работать с внешними API, и как приятно видеть, что твой продукт может быть настоящим приложением, полезным фанатам аниме.

ЯНА

А я поняла, как важно продумывать пользовательский опыт заранее, как интересно создавать интуитивные интерфейсы и работать с Material Design 3, и как приятно видеть, что твой дизайн может быть настоящим приложением, которое нравится пользователям и помогает им получать удовольствие от коллекционирования персонажей.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

МИША

Спасибо за внимание! Готовы ответить на ваши вопросы — по архитектуре, технологиям, реализации, тестированию или любым другим аспектам проекта.

# Возможные вопросы на защите и ответы

Вопросы по проекту:

Как реализовано получение данных о персонажах и аниме?

○ Ответ: Данные о персонажах хранятся локально в Room, информация об аниме получается через Jikan API.

Как работает система кристаллов?

○ Ответ: Кристаллы начисляются за просмотр рекламы, тратятся на открытие лутбоксов, баланс сохраняется в Room.

Как реализована система редкости персонажей?

○ Ответ: Используется алгоритм с процентами: обычные 70%, редкие 25%, легендарные 5%.

Как происходит поиск аниме?

○ Ответ: Через Jikan API (MyAnimeList), используем Retrofit для HTTP-запросов.

Как реализована темная тема?

○ Ответ: Используем Material Design 3, переключатель в настройках, состояние сохраняется в SharedPreferences.

Как устроены мок-данные и почему вы их используете?

○ Ответ: Мок-данные — это заранее подготовленные списки персонажей из популярных аниме. Мы используем их, чтобы приложение работало полноценно даже без внешнего API для персонажей.

Как тестировали офлайн-режим?

○ Ответ: Отключали интернет на устройстве/эмуляторе и проверяли, что данные из Room отображаются корректно, а приложение не падает.

## Вопросы по технологиям:

Что такое Jetpack Compose?

○ Ответ: Современный UI toolkit для Android, декларативный подход к созданию интерфейса.

Как работает Room?

○ Ответ: ORM для SQLite, позволяет хранить объекты и получать их через DAO.

Для чего нужен Hilt?

○ Ответ: Dependency Injection, автоматическое создание и внедрение зависимостей.

Как реализована асинхронность?

○ Ответ: Coroutines и Flow для асинхронных операций и реактивных потоков данных.

Как устроен жизненный цикл Composable?

○ Ответ: Composable-функции пересоздаются при изменении состояния (рекомпозиции), важно использовать remember и rememberSaveable для сохранения данных.

Как реализована навигация между экранами?

○ Ответ: Navigation Compose, NavController, маршруты вынесены в отдельный файл. Параметры передаются через аргументы.

Как реализовано хранение данных?

○ Ответ: Room для локальных данных, Jikan API для поиска аниме, SharedPreferences для настроек.

Какие паттерны проектирования использовали?

○ Ответ: MVVM, Repository, Singleton, Clean Architecture.

Как бы реализовали push-уведомления?

○ Ответ: В интернете рекомендуют Cloud Messaging, думаю полученных знаний хватает разобраться в настройке.

Как бы добавили аналитику?

○ Ответ: Интегрировали бы Amplitude или Яндекс.Метрику, отправляли бы события при открытии лутбоксов, поиске аниме.

Почему не реализованы некоторые функции?

○ Ответ: Не хватило времени и опыта, часть функций (push-уведомления, звуковые эффекты) оставили на будущее, чтобы не усложнять проект.

## Совет:

Если не знаете точный ответ — честно скажите, что делали по примеру из курса, не до конца разобрались, но готовы изучить глубже. Попробуйте рассказать общую идею и ваше понимание.