Добрый день, уважаемая комиссия!

Я Артюшкина Ирина Александровна, экономический факультет представляю проект мобильного приложения CR.

Это результат нескольких месяцев изучения Android-разработки, обсуждений, проб и ошибок. Я горжусь тем, что получилось реализовать современное приложение для ресторанного бизнеса, максимально приближенное к реальным задачам.

**0. Как я работала над проектом**

Над проектом я работала самостоятельно, поэтому мне пришлось совмещать сразу несколько ролей.

Я отвечала за все этапы разработки: писала код, разбиралась с архитектурой, интегрировала библиотеки, настраивала навигацию, работала с базой данных и backend-частью.

Кроме технической реализации, я самостоятельно занималась анализом рынка, сбором требований, общением с потенциальными пользователями, а также планированием задач, оформлением документации, подготовкой презентации и отчёта по проекту.

Когда возникали сложные моменты — например, при работе с DI или Room, — я искала решения в документации, смотрела обучающие видео, читала статьи и форумы.

В процессе работы я освоила новые для себя технологии, научилась быстро разбираться в сложных темах и принимать решения самостоятельно.

Хотя по специальности я не являюсь программистом, этот проект помог мне развить как технические, так и организационные навыки, а также научил работать с современными инструментами командной и проектной работы.

1. **Введение и мотивация**

Выбор темы был свободным — можно было выбрать из предложенных вариантов или предложить свою идею.

Я решила создать мобильное приложение для ресторана, потому что сама часто сталкивалась с тем, что не во всех кафе/ресторанах есть такое приложение: приходится звонить, писать в мессенджеры, иногда заказы теряются или возникают путаницы с меню и временем.

Мне показалось, что современное мобильное приложение может сделать этот процесс гораздо удобнее — как для клиентов, так и для сотрудников ресторана.

Я хотела реализовать решение, которое позволит быстро просматривать меню, оформлять заказы, отслеживать их статус, получать уведомления и управлять своим профилем — всё в одном приложении.

**2. Исследование рынка**

Объем рынка ресторанного бизнеса в РФ ~2.4 трлн. рублей.

В 2024 г. объем онлайн-продаж продуктов питания достиг 1,2 трлн. рублей.

В 2024 г. количество онлайн заказов - 781 млн.

В России около 70% населения предпочитают пользуются услугами доставки готовой еды.

**3. Минусы работы ресторанов без приложения**

* Упущенные возможности для продаж
* Проблемы с взаимодействием с клиентами
* Проблемы с оптимизацией процессов

**4. Плюсы работы ресторанов с приложением**

* Повышение лояльности клиентов
* Улучшение коммуникации
* Удобство для доставки
* Оптимизация работы персонала
* Удобство заказа и оплаты

**5. Основной функционал и пользовательский сценарий**

Я старалась, чтобы каждый экран приложения решал конкретную задачу пользователя и был максимально удобен. В результате реализованы следующие экраны:

Аутентификация пользователей – это регистрация и вход в аккаунт, а также безопасное хранение данных пользователей в локальной базе данных

Меню — чтобы клиент мог быстро найти нужное блюдо, цену и выбрать подходящее.

Категории блюд — для удобной навигации по меню (завтраки, горячие блюда, салаты, десерты, напитки и т.д.).

Корзина — для формирования заказа, редактирования позиций и оформления покупки.

Профиль пользователя — для просмотра и редактирования личных данных, а также отслеживания баллов лояльности.

Система уведомлений — чтобы клиент всегда был в курсе статуса заказа.

Каждый экран я проектировала, исходя из реальных сценариев использования.

Моя цель была — сделать приложение не только современным, но и действительно полезным для клиентов ресторана.

**6. Демонстрация экранов**

(В этот момент я показываю презентацию или приложение в эмуляторе.)

CR – это мобильное приложение для Android, разработанное с использованием Jetpack Compose и Material 3, предназначенное для управления ресторанным сервисом.

На слайде представлены экраны:

* WelcomeScreen — приветственный экран с фирменным фоном, крупной надписью "Добро пожаловать!" и двумя большими кнопками: "Войти" и "Зарегистрироваться". Всё выполнено в фирменных цветах.
* LoginScreen — экран входа с полями для email и пароля, стилизованными под общий дизайн приложения. Используются кастомные кнопки (CRButton), а при успешном входе появляется всплывающее сообщение «Вход выполнен успешно».
* RegisterScreen — экран регистрации с полями для имени, email и пароля. Есть обработка ошибок (например, если пользователь уже существует), индикатор загрузки при регистрации, а также кнопки для возврата назад и очистки базы данных.
* FirstScreen — главный экран после входа. Здесь отображается приветствие с именем пользователя, слайдер с фотографиями ресторана, описание атмосферы, а также горизонтальная панель быстрых кнопок для перехода на основные разделы: Главная, Меню, Корзина, Профиль. Удобная навигация реализована через боковое меню и нижнюю панель.
* MenuScreen — экран меню с карточками категорий ("Завтраки", "Салаты", "Горячее", "Десерты", "Напитки", "Закуски"). Каждая карточка содержит изображение и название категории, при нажатии происходит переход к соответствующему разделу.
* BreakfastScreen, SaladsScreen, HotDishesScreen, DessertsScreen, DrinksScreen, SpecialsScreen — экраны с блюдами соответствующих категорий.

Каждый экран оформлен в едином стиле: заголовок, сетка карточек с фото блюда, названием, ценой и кнопкой "Добавить в корзину".

Карточки аккуратно выровнены, изображения занимают большую часть карточки, кнопки стилизованы под фирменные цвета.

* CartScreen — экран корзины.

Здесь отображаются все добавленные блюда с фото, названием, ценой и возможностью изменить количество или удалить позицию.

Внизу — итоговая сумма и большая кнопка "Заказать".

После оформления заказа появляется уведомление о сумме и начисленных баллах, корзина очищается.

* ProfileScreen — экран профиля пользователя.

Отображается фото профиля (с возможностью загрузки нового), имя пользователя (с возможностью редактирования), количество накопленных баллов.

Есть кнопки для обращения в поддержку и просмотра информации о сервисе.

Реализованы всплывающие диалоги для редактирования имени, подтверждения загрузки фото, а также подробная информация о баллах.

Особенности оформления:

* Все экраны используют фирменный фон (HeartBackground), плавные отступы, адаптивную верстку.
* Кнопки и поля ввода стилизованы в едином стиле, используются кастомные компоненты (CRButton).
* Для изображений блюд и профиля используется библиотека Coil.
* Реализована поддержка вертикальной прокрутки для длинных списков.
* Везде, где это возможно, реализована обратная связь для пользователя: всплывающие сообщения, индикаторы загрузки, диалоги подтверждения.

Все экраны оформлены так, чтобы пользователь мог понять логику работы приложения и получить приятный визуальный опыт.

**7. Дизайн**

Основной цвет:

* светлый, нежно-розовый оттенок, который используется для фона верхней панели, нижней панели и бокового меню.

Цвет текста и иконок:

* Текст и иконки на панелях — чёрный, что обеспечивает хороший контраст с основным цветом.
* Текст в боковом меню — белый, для контраста с розовым фоном.

**8. Архитектура и структура проекта**

Я выбрала архитектуру MVVM с использованием Jetpack Compose.

Честно говоря, изначально я думала, что можно реализовать всё через простой подход с одной или несколькими MainActivity, но по мере изучения и реализации стало понятно, что для приложения такого уровня нужно чётко разделять логику на компоненты.

В проекте реализованы следующие слои:

Presentation Layer — все экраны построены на Jetpack Compose. Каждый экран — это отдельный Composable, а логика работы экранов частично вынесена во ViewModel. Такой подход позволяет легко управлять состоянием UI и переиспользовать компоненты.

Data Layer — слой работы с данными. Здесь используется Room для локального хранения информации о пользователях, заказах и корзине. DAO (Data Access Object) обеспечивает удобный доступ к базе данных.

Бизнес-логика в основном находится в UI слое, что упрощает понимание и поддержку кода для учебного проекта. В будущем планирую изучить и внедрить Clean Architecture для лучшего разделения ответственности между слоями.

**9. Реализация: детали и этапы**

* Весь UI построен на Jetpack Compose с использованием Material Design 3.

Например, экран меню реализован с помощью вертикального списка (LazyColumn), где каждая карточка блюда — это отдельный Composable-элемент с фото, названием, ценой и кнопкой "Добавить в корзину".

* Навигация реализована через Navigation Compose, все маршруты вынесены в отдельный файл Routes.kt для удобства поддержки и масштабирования.
* Для хранения данных о пользователях и заказах используется Room.

Это позволяет хранить всю информацию локально и работать с приложением даже без интернета. Данные о блюдах пока хранятся в виде мок-объектов прямо в коде.

Для отображения изображений используется painterResource для локальных изображений блюд и Coil для загрузки изображений профиля пользователя.

Push-уведомления реализованы через системные уведомления Android.

После оформления заказа пользователь получает настоящее push-уведомление с информацией о сумме заказа, начисленных баллах и времени доставки. Для этого используется NotificationManager и создаётся отдельный канал уведомлений для заказов.

Часть функций работает на мок-данных — например, меню блюд и история заказов берутся из заранее подготовленных списков. Это позволяет показать логику работы приложения даже без подключения к серверу.

**10. Почему именно такой стек**

Я выбрала этот стек, потому что он современный, поддерживается Google и широко используется в реальных Android-проектах.

* Jetpack Compose — позволяет быстро создавать адаптивный и красивый интерфейс, упрощает работу с состоянием UI.
* Material Design 3 — обеспечивает единый стиль и современный внешний вид приложения.
* MVVM — разделяет логику, данные и интерфейс, делает проект удобным для поддержки и развития.
* Room — надёжное локальное хранение данных, поддержка офлайн-режима.
* Kotlin Coroutines — удобная и современная работа с асинхронностью.
* Navigation Compose — простая и гибкая навигация между экранами.
* Coil — быстрая загрузка и кэширование изображений профиля пользователя.
* Системные уведомления — пользователь всегда в курсе важных событий.

Связи между слоями реализованы вручную, чтобы лучше понять архитектуру. DI-фреймворки и облачные сервисы планирую изучить и внедрить в будущем. Такой стек позволил сделать приложение современным, удобным и готовым к развитию.

**11. Где и как использованы технологии**

* Jetpack Compose

Использован для построения всего пользовательского интерфейса: все экраны (Welcome, Login, Register, Меню, Корзина, Профиль и др.) реализованы как отдельные Composable-функции.

* Material Design 3

Применён для оформления всех элементов интерфейса: кнопки, поля ввода, карточки блюд и категорий, навигационные панели — всё выполнено в едином стиле.

* MVVM

Логика работы экранов частично вынесена во ViewModel, что упростило поддержку и тестирование приложения.

* Room

Используется для локального хранения данных о пользователях, заказах и корзине. Благодаря этому приложение работает даже без интернета. Данные о блюдах и меню реализованы как мок-объекты.

* Kotlin Coroutines

Применяются для асинхронной загрузки и обновления данных, чтобы интерфейс всегда оставался отзывчивым.

* Navigation Compose

Реализует переходы между экранами, маршруты вынесены в отдельный файл для удобства поддержки.

* Coil

Используется для загрузки и отображения изображений профиля пользователя, с поддержкой кэширования.

* Системные уведомления

После оформления заказа пользователю приходит push-уведомление с деталями заказа и начисленными баллами.

**12. База данных**

Таблица: users

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| id | String | Первичный ключ |
| name | String | Имя пользователя |
| email | String | Email пользователя |
| password | String | Пароль |
| profileImageUri | String? | URI изображения профиля |
| points | Int | Баллы пользователя |

DAO (UserDao):

* Вставка пользователя (insert)
* Получение пользователя по email/id
* Удаление всех пользователей
* Обновление изображения профиля
* Обновление баллов
* Получение баллов по id

AppDatabase:

* Содержит только одну сущность: User
* Предоставляет доступ к UserDao

**13. Трудности и решения**

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

* Room — сначала возникали трудности с созданием сущностей и связей между ними, а также с миграциями базы данных. Для тестирования использовала мок-данные, чтобы не потерять информацию при ошибках миграции. Постепенно разобралась, как правильно строить структуру БД и проводить миграции.
* Асинхронность и Coroutines — пришлось разбираться с корутинами, чтобы не блокировать основной поток при работе с базой данных и обновлении UI. Иногда было сложно понять, почему данные не обновляются сразу, приходилось читать документацию и искать решения на форумах.
* Push-уведомления — удалось реализовать настоящие системные уведомления через NotificationManager, чтобы пользователь получал сообщения о заказах. Однако интеграцию с облачными push-сервисами (например, Firebase Cloud Messaging) пока не реализовала. Планирую изучить этот вопрос и внедрить полноценные push-уведомления в будущем.
* Внедрение зависимостей (DI) — в этом проекте зависимости между слоями реализованы вручную. DI-фреймворки, такие как Hilt или Koin, пока не использовала, чтобы лучше понять архитектуру приложения. В дальнейшем хочу изучить эти инструменты и внедрить их для упрощения поддержки и тестирования кода.
* Синхронизация с облаком — полноценная синхронизация данных с сервером пока не реализована, часть функций работает на мок-данных. В будущем планирую интегрировать облачные сервисы для расширения функциональности и поддержки real-time обновлений.

**14. Преимущества решения**

К ним относятся:

1. Современные технологии: Jetpack Compose и Material Design 3

2. Offline-first подход (работа без подключения к интернету)

3. Масштабируемая архитектура: Clean Architecture + MVVM

4. Real-time обновления (мгновенная синхронизация данных)

5. Интуитивный UX/UI (понятный интерфейс для всех пользователей)

6. Безопасность и приватность

7. Готовность к масштабированию

**15. Перспективы развития**

В будущем планирую:

1. Онлайн-бронирование столиков — Выбор зала, стола и времени посещения прямо в приложении.
2. iOS-версия приложения — Разработка кроссплатформенного решения с помощью Compose Multiplatform или нативного Swift.
3. Интеграция с платежными системами — Онлайн-оплата заказов, чаевые, поддержка Apple Pay, Google Pay и других сервисов.
4. Программа лояльности и бонусы — Накопление баллов, персональные скидки, подарки ко дню рождения, уровни лояльности.
5. Push-уведомления, персонализированные предложения и напоминания — Автоматические уведомления о статусе заказа, акциях, специальных предложениях и напоминания о бронированиях.
6. Интеграция с доставкой — Оформление и отслеживание доставки блюд, интеграция с агрегаторами (Яндекс.Еда, Delivery Club и др.), отслеживание курьера в реальном времени.
7. Социальные функции — Отзывы и рейтинги, возможность делиться заказами и блюдами в соцсетях, приглашения друзей, интеграция с мессенджерами (чат с рестораном, поддержка).
8. Аналитика для владельцев и управляющих — Дашборды, отчёты по продажам, популярности блюд, активности пользователей, аналитика по филиалам.
9. Видеоконсультации с шеф-поваром или сомелье — Онлайн-мероприятия, мастер-классы, прямые эфиры для клиентов.
10. Административная панель — Веб-интерфейс для управления меню, заказами, бронированиями, акциями и аналитикой.
11. Интеграция с внешними сервисами — Карты (Google/Yandex), CRM, системы учёта склада, email-рассылки.
12. Мультибрендовость и поддержка филиалов — Возможность работы с несколькими ресторанами/филиалами в одном приложении.
13. API для партнёров и корпоративных клиентов — Открытый API для интеграции с корпоративными заказчиками и сторонними сервисами.
14. Доступность и инклюзивность — Поддержка скринридеров, крупного шрифта, адаптация для людей с ограниченными возможностями.

**16. Личный опыт и выводы**

Над проектом я работала самостоятельно, поэтому мне пришлось совмещать сразу несколько ролей: разработчика, аналитика, дизайнера и тестировщика. За время работы я не только научилась программировать и разбираться в современных технологиях Android, но и планировать задачи, анализировать требования, оформлять результаты и презентовать проект.Я особенно поняла, как важно продумывать архитектуру заранее, чтобы приложение было удобным для поддержки и развития. Было сложно, но очень интересно реализовывать офлайн-режим, работу с базой данных и push-уведомления.Самое приятное — видеть, что твой продукт может быть реально полезен людям и решать их повседневные задачи.Если бы было больше времени, я бы обязательно доработала полноценную синхронизацию с облаком, расширила систему уведомлений и интегрировала онлайн-оплату. Очень хочется продолжать развиваться в мобильной разработке и внедрять новые технологии в будущих проектах.

Спасибо за внимание! Готова ответить на ваши вопросы

### **Вопросы по проекту:**

* Как реализовано получение данных о клиентах, услугах, мастерах?
  + Ответ: Основные данные (услуги, мастера, акции) сейчас берутся из Supabase удаленной базы данных, на случай недостающих данных подготовлены массивы из мок-объектов — это заранее подготовленные списки в коде.
  + Мы сделали так, потому что не успели реализовать полноценный backend. Для профиля пользователя планировали использовать авторизацию supabase authentication, но пока работает только локально через Room.
* Как осуществляется кэширование?
  + Ответ: Для кэширования и офлайн-режима используем Room. Все основные сущности (услуги, мастера, бронирования) сохраняются в локальной базе. Если нет интернета — показываем данные из Room или из мок-объектов.
* Как пользователь может добавить запись или изменить данные?
  + Ответ: Добавление и изменение реализовано в ручную через редактирования таблицы supabase, авторизацию планировали supabase authentication но пока не успели подключить — данные сохраняются в Room, а при появлении интернета (в будущем) будет синхронизация с Supabase.
* Как реализована работа с тёмной темой?
  + Ответ: Используем Material 3 и Jetpack Compose, переключатель темы есть в настройках, состояние темы сохраняется в SharedPreferences.
* Как происходит взаимодействие с внешним API?
  + Ответ: В нашем проекте внешний API — только Supabase, и то только для синхронизации данных пользователя мастера, услуги и бронирований. Все остальные данные (отзывы, лента новостей, профиль пользователя) — это мок-данные.
* Почему не реализованы push-уведомления?
  + Ответ: Не хватило времени и опыта работы с Cloud Messaging. Сейчас уведомления реализованы как анимация, но планируем доработать.
* Как устроены мок-данные и почему вы их используете?
  + Ответ: Мок-данные — это обычные списки объектов, которые мы храним прямо в коде (например, список мастеров, услуг, акций).
  + Мы используем их, чтобы приложение работало и выглядело полноценно даже без полноценного backendа и внешнего API. Мы думали использовать внешние данные через API но не нашли подходящие данные с открытым доступом для салона красоты.
* Как тестировали офлайн-режим?
  + Ответ: Отключали интернет на устройстве/эмуляторе и проверяли, что данные из Room и мок-объектов отображаются корректно, а приложение не падает.
* Как бы реализовали push-уведомления?
  + Ответ: В интернете рекомендуют Cloud Messaging, думаю полученных знаний хватает разобраться в настройке.
* Как бы реализовали интеграцию с платежами?
  + Ответ: Рассматривали бы Яндекс.Кассу (через SDK) или Telegram Pay (через Bot API), реализовали бы отдельный экран оплаты и обработку статуса платежа.
* Как бы добавили аналитику?
  + Ответ: Интегрировали бы Amplitude или Яндекс.Метрику, отправляли бы события при бронировании, оплате, просмотре экрана.
* Как бы реализовали поддержку нескольких салонов?
  + Ответ: Самое простое решение в упор создать отдельный клон приложения с отдельной бд.
  + Если у нас салон относится к одной франшизе, тогда добавили бы сущность "Салон" в базу, реализовали бы выбор салона на первом экране или на новом экране, фильтрацию услуг и мастеров по выбранному салону.

**Вопросы по темам курса:**

* Что такое activity
  + Activity в Android — компонент приложения, который представляет собой отдельный экран или окно. Каждая Activity обычно соответствует одному экрану пользовательского интерфейса или фрагменту функциональности.
* Как устроен жизненный цикл Activity?
  + Ответ: Activity проходит стадии onCreate, onStart, onResume, onPause, onStop, onDestroy. Важно правильно управлять ресурсами и состоянием.
* 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: Supabase для удаленных данных,Room для сложных структур, SharedPreferences для настроек, мок-данные для демонстрации.
* Как работает Room и для чего он используется?
  + Ответ: Room — это библиотека для работы с SQLite, позволяет хранить объекты и получать их через DAO. В нашем проекте — для бронирований, профиля, истории.
* Как внедряются зависимости через Hilt?
  + Ответ: Используем аннотации @Inject, @HiltViewModel, модули для предоставления зависимостей. Честно: делал по примеру из курса, до конца не разобрался, насколько я понял это облегчает тестирование и поддержку кода.
* Как тестировали приложение?
  + Ответ: Вручную, на эмуляторе и устройстве, тесты есть но простые не сильно вникали в данную тему в первую очередь старались максимально реализовать функциональную часть приложения, но структура позволяет их добавить.
* Как бы реализовали поддержку нескольких языков?
  + Ответ: Через ресурсы strings.xml, локализацию, переключение языка в настройках.
* Какие трудности были с архитектурой?
  + Ответ: Сложно было понять, как правильно разделять слои и для чего это используется,

Почему не реализованы некоторые функции?

* + Ответ: Не хватило времени и опыта, часть функций (push, оплата, полноценный backend) оставили на будущее, чтобы не усложнять проект, попытаться реализовать большую часть и не делать неработающие заглушки.

**Вопросы по Android-разработке:**

* Чем отличается ViewModel от Activity?
  + Ответ: ViewModel хранит состояние и логику, переживает повороты экрана, Activity — только UI.
* Для чего нужен Hilt и как он работает?
  + Ответ: Для автоматического внедрения зависимостей, чтобы не создавать их вручную. Работает через аннотации и генерирует код для DI.
* Как реализовать асинхронную загрузку данных?
  + Ответ: Coroutines, suspend-функции, Flow.
* Как реализовать сохранение состояния при повороте экрана?
  + Ответ: ViewModel, rememberSaveable, SavedStateHandle.
* Какие паттерны проектирования использовали?
  + Ответ: MVVM, Repository, Singleton