Министерство образования и науки Нижегородской области

Государственное бюджетное профессиональное образовательное учреждение

**«Нижегородский Губернский колледж»**

Методическая комиссия Информатика и вычислительная техника

Допущен к защите:

преподаватель

\_\_\_\_\_\_\_\_\_\_Ю. С. Мамшева

« » 2024 г.

РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ

**ПРОЕКТНАЯ РАБОТА**

по МДК 01.03 «Разработка мобильных приложений»

Студент: \_\_\_\_\_\_\_\_\_\_\_\_ Е.С.Колиниченко 27.11.2024

(подпись)

Специальность, группа: 09.02.07, 43П

Нижний Новгород

2024

СОДЕРЖАНИЕ

# ГЛАВА 1. ПОСТАНОВКА ЦЕЛЕЙ И ЗАДАЧ

1. **Техническое задание**
   1. **Общие сведения**
      1. Наименование системы

Полное наименование: Meow Mates

Краткое наименование: MM.

* + 1. Наименование организаций

Заказчик: Мамшева Юлия Сергеевна

Разработчик: Колиниченко Екатерина Сергеевна

* + 1. Плановые сроки начала и окончания работы

Начало работ: 2 октября 2024

Окончание работ: запланировано на момент выполнения всех стадий разработки, что включает 25 очных пар по 1,5 часа каждая. Таким образом, окончание работ будет определено по завершении всех запланированных занятий.

* + 1. Порядок оформления и предъявления заказчику результатов работ

Работы по созданию мобильного приложения Meow Mates сдаются Разработчиком поэтапно в соответствии с календарным планом. По окончании каждого этапа Разработчик предоставляет Заказчику соответствующую часть отчетных документов, ссылки на репозитории, а также демонстрирует результаты, разработанные в рамках конкретного этапа, которые прописаны и согласованы в документе с Заказчиком.

* 1. **Назначение и цели создания системы**
     1. Актуальность

В современном мире наблюдается рост интереса к домашним животным, особенно к кошкам. Владельцы все чаще ищут способы социализации своих питомцев и создания для них дружеских связей. Мобильное приложение Meow Mates предоставляет уникальную платформу, где владельцы могут зарегистрировать своих пушистиков и находить подходящих "партнеров" для их котиков. Это не только способствует улучшению эмоционального состояния животных, но и создает сообщество любителей кошек, где они могут обмениваться опытом и советами.

Таким образом, разработка Meow Mates актуальна как для владельцев кошек, так и для их питомцев, стремящихся к социальной активности.

* + 1. Цели и задачи

Цель: Создание мобильного приложения, предназначенного для помощи владельцам кошек в поиске подходящих "партнеров" для их питомцев.

Задачи, которые можно выделить в контексте данной цели:

1. Создание репозитория на СКВ
2. Разработка интерфейса приложения для удобной регистрации питомцев.
3. Создание системы профилей для кошек с возможностью добавления фотографий и описаний.
4. Реализация алгоритма поиска и сопоставления питомцев по интересам и характеристикам.
5. Внедрение системы сообщений для общения между владельцами.
6. Проектирование и реализация серверной части приложения.
7. Тестирование серверной части приложения.
8. Обеспечение безопасности данных пользователей.
9. Разработка документации по использованию приложения.
   1. **Характеристика объектов автоматизации**

Заказчиком является компания, специализирующаяся на разработке приложений для домашних животных.

Объекты автоматизации в приложении Meow Mates могут включать в себя следующие элементы:

1. Интерфейс пользователя.

2. Модули регистрации и поиска питомцев.

3. Система управления профилями пользователей.

* 1. **Требования к системе**
     1. Требования к системе в целом
        1. Требования к структуре и функционированию системы

Система Meow Mates должна быть централизованной, т.е. все данные должны располагаться в центральном хранилище, использующем Supabase.

Система должна быть построена на двух уровнях: владелец питомца и питомец (кот).

Источниками данных для системы должны быть:

* Supabase как облачная база данных для хранения информации о пользователях и их питомцах.

Система должна поддерживать:

* работу пользователей в режиме – 24 часов в день, 7 дней в неделю (24х7);
* выполнение своих функций – регистрацию пользователей, регистрацию питомцев, поиск пар;
* хранение данных, предоставление отчетности.

Для обеспечения высокой надежности функционирования системы как в целом, так и её отдельных компонентов должно обеспечиваться выполнение требований по диагностированию её состояния.

Диагностирование системы должно осуществляться с использованием встроенных средств мониторинга, предоставляемых Supabase.

* + - 1. Требования к численности и квалификации персонала системы и режиму его работы

В состав персонала, необходимого для обеспечения эксплуатации Meow Mates, необходимо выделение следующих ответственных лиц:

* Главный администратор – 1 человек.
* Технические специалисты для поддержки приложения.

К квалификации персонала, эксплуатирующего систему, предъявляются следующие требования:

* Конечный пользователь — знание основ работы с мобильными приложениями и умение пользоваться смартфоном.
  + - 1. Показатели назначения

В зависимости от различных вероятных условий система должна выполнять требования, приведенные в таблице 1.

Таблица 1 – Требования к сохранению работоспособности системы

|  |  |
| --- | --- |
| Вероятное условие | Требование |
| Нарушения в работе системы внешнего электроснабжения серверного оборудования продолжительностью до 15 мин. | Функционирование в полном объеме. |
| Нарушения в работе системы внешнего электроснабжения серверного оборудования продолжительностью до 5 ч. | Аварийное функционирование системы некоторое время, сохранение всех данных. |
| Выход из строя сервера подсистемы хранения данных. | Уведомление администратора о проблеме и автоматическое восстановление данных из резервной копии. |
| Уязвимость системы из-за устаревшего ПО. | Регулярное обновление программного обеспечения и библиотек приложений. |

* + - 1. Требования к надежности

Уровень надежности мобильного приложения Meow Mates должен достигаться согласованным применением организационных, организационно-технических мероприятий и программных средств.

Надежность должна обеспечиваться за счет:

* регулярного обновления программного обеспечения;
* резервного копирования данных в облачном хранилище (Supabase);
* мониторинга работы приложения и его компонентов;
* тестирования на устойчивость к сбоям и ошибкам.

Время устранения отказа должно быть следующим:

* при перерыве и выходе за установленные пределы параметров электропитания — не более 150 минут;
* при сбоях в работе приложения — не более 24 часов.

Система должна соответствовать следующим параметрам:

* среднее время восстановления 2 часов;
* время наработки на отказ 1 час.

Перечень аварийных ситуаций, по которым регламентируются требования к надежности

При работе системы возможны следующие аварийные ситуации, которые влияют на надежность работы системы:

* сбой в работе сервера Supabase;
* сбой в работе мобильного устройства пользователя;
* ошибки в коде приложения, не выявленные при тестировании;
* проблемы с сетью, влияющие на доступ к облачным данным.

Требования к надежности технических средств и программного обеспечения:

К надежности мобильного приложения предъявляются следующие требования:

* приложение должно быть разработано с учетом принципов устойчивости к сбоям;
* необходимо проводить регулярные обновления для устранения уязвимостей и повышения стабильности работы;
* приложение должно иметь встроенные механизмы для автоматического восстановления данных после сбоев.

Эти требования обеспечат высокую степень надежности и стабильности работы мобильного приложения Meow Mates, что является критически важным для удовлетворения потребностей пользователей.

* + - 1. Требования к эргономике и технической эстетике

Подсистема должна обеспечивать удобный для конечного пользователя интерфейс, отвечающий следующим требованиям.

В части внешнего оформления:

* интерфейсы подсистем должен быть типизированы;
* должно быть обеспечено наличие русскоязычного интерфейса пользователя;
* должен использоваться понятный шрифт;
* размер шрифта должен быть: 20 пт;
* цветовая палитра должна быть привлекательной и гармоничной.

В части диалога с пользователем:

* при возникновении ошибок в работе подсистемы должно выводиться сообщение с наименованием ошибки и с рекомендациями по её устранению на русском языке.
  + - 1. Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы

Условия эксплуатации, а также виды и периодичность обслуживания мобильного приложения Meow Mates должны соответствовать требованиям по эксплуатации, техническому обслуживанию и ремонту, изложенным в документации разработчика.

Приложение должно быть доступно для пользователей на мобильных устройствах с операционными системами Android.

Для обеспечения надежности работы приложения необходимо:

Регулярно обновлять приложение для устранения уязвимостей и добавления новых функций.

Проводить тестирование приложения на различных устройствах для обеспечения совместимости и стабильности работы.

Обеспечивать резервное копирование данных пользователей в облачном хранилище (Supabase) для предотвращения потери информации.

Состав, место и условия хранения данных пользователей определяются в соответствии с политикой конфиденциальности и правилами использования приложения.

Для обеспечения бесперебойной работы приложения необходимо следить за состоянием серверов Supabase, которые обеспечивают хранение данных. В случае возникновения сбоев или проблем с доступом к серверу, пользователи должны получать уведомления о статусе работы приложения через интерфейс.

Техническое обслуживание приложения включает в себя:

* Мониторинг производительности и отзывчивости приложения.
* Обновление библиотек и зависимостей, используемых в приложении.
* Проведение регулярных проверок безопасности для защиты данных пользователей от несанкционированного доступа.

Все действия по техническому обслуживанию и обновлению приложения должны документироваться для обеспечения прозрачности процессов и возможности анализа в случае возникновения проблем.

* + - 1. Требования к защите информации от несанкционированного доступа

Обеспечение информационной безопасности системы Meow Mates должно удовлетворять следующим требованиям:

Защита системы должна обеспечиваться комплексом программно-технических средств и поддерживающих их организационных мер.

Защита системы должна обеспечиваться на всех технологических этапах обработки информации и во всех режимах функционирования, включая ремонтные работы.

Программно-технические средства защиты не должны существенно ухудшать основные функциональные характеристики системы (надежность, быстродействие).

Разграничение прав доступа пользователей должно строиться по принципу "что не разрешено, то запрещено".

Средства антивирусной защиты должны быть установлены на всех рабочих местах пользователей и администраторов системы Meow Mates и должны обеспечивать:

* централизованное управление сканированием;
* централизованную автоматическую инсталляцию клиентского ПО;
* централизованное автоматическое обновление вирусных сигнатур;
* ведение журналов вирусной активности;
* администрирование всех антивирусных продуктов.
  + - 1. Требования к защите от влияния внешних воздействий

Применительно к программно-аппаратному окружению Системы предъявляются следующие требования к защите от влияния внешних воздействий.

Требования по стойкости, устойчивости и прочности к внешним воздействиям: Система должна иметь возможность функционирования при колебаниях напряжения электропитания в пределах от 155 до 265 В (220 ± 20 % - 30 %);

* + - 1. Требования по стандартизации и унификации

Разработка приложения должна осуществляться с использованием стандартных методологий функционального моделирования: IDEF0, DFD и информационного моделирования IE в рамках рекомендаций по стандартизации.

Для разработки пользовательских интерфейсов должны использоваться возможности Jetpack Compose для Kotlin.

* + - 1. Дополнительные требования

Приложение должно разрабатываться и эксплуатироваться на уже имеющемся у Заказчика аппаратно-техническом комплексе (смартфоны). Необходимо создать отдельные зоны разработки и тестирования приложения Meow Mates.

* + - 1. Требования безопасности

При внедрении, эксплуатации и обслуживании технических средств приложения должны выполняться меры электробезопасности в соответствии с действующими нормами безопасности.

Аппаратное обеспечение приложения должно соответствовать требованиям пожарной безопасности в производственных помещениях согласно актуальным стандартам.

Должно быть обеспечено соблюдение общих требований безопасности при обслуживании приложения в процессе эксплуатации.

* + 1. Требования к функциям, выполняемым системой

1. Авторизация и регистрация, выход из системы:

* Регистрация в системе с проверкой на корректность введенных данных;
* Авторизация в системе с хэшированием пароля
* Возможность выхода из системы

1. Регистрация питомцев в системе:

* Возможность добавления нового котика с указанием его характеристик (имя, порода, возраст, фото).
* Возможность редактирования информации о питомце.
* Возможность удаления питомца из профиля.

1. Просмотр и поиск котиков

* Возможность просмотра списка котиков поблизости.
* Возможность сортировки и фильтрации котиков по различным параметрам (возраст, порода, пол).
* Возможность просмотра профиля других котиков с их характеристиками.

1. Просмотр профиля владельца:

* Возможность редактирования информации о владельце (имя, контактные данные).
* Просмотр статистики взаимодействия с другими пользователями (например, количество добавленных питомцев, количество просмотров профиля).
  + 1. Требования к видам обеспечения
       1. Требования к информационному обеспечению

База данных должна обеспечивать надежное хранение информации о пользователях и их питомцах, а также статистику взаимодействия между ними. Интерфейс приложения должен быть интуитивно понятным и удобным для пользователей, что позволит им быстро освоить работу с системой.

* + - 1. Требования к аппаратному обеспечению

Мобильные устройства (смартфоны) с достаточной производительностью для работы приложения Meow Mates. Сервер для хранения данных в облаке (Supabase) и обеспечения доступа к системе через интернет.

* + - 1. Требования к программному обеспечению

Специализированное программное обеспечение для автоматизации процесса регистрации питомцев и их поиска, а также для управления данными пользователей и питомцев.

* + - 1. Требования к контролю, хранению, обновлению и восстановлению данных

Система должна протоколировать все события, связанные с изменением информационного наполнения, и иметь возможность восстанавливать свое состояние в случае сбоя, используя ранее запротоколированные изменения данных. Хранение исторических данных должно производиться не более чем за 5 (пять) предыдущих лет; по истечении данного срока данные должны переходить в архив.

* + - 1. Требования к методическим материалам

Подготовка подробных инструкций по работе с мобильным приложением для пользователей различного уровня подготовки, включая руководство по установке и настройке приложения Meow Mates.

Эти требования обеспечивают функциональность и надежность мобильного приложения Meow Mates, а также удобство использования для владельцев кошек при поиске подходящих "партнеров" для своих питомцев.

* 1. **Стадии разработки системы, ресурсы, роли**



Основные стадии разработки системы:

1. Проектирование целей и задач;
2. Разработка макета приложения;
3. Разработка серверной части приложения;
4. Тестирование серверной части приложения;
5. Разработка мобильного приложения;
6. Тестирование мобильного приложения;

В качестве среды разработки была выбрана Android Studio. Приложение разрабатывается на языке программирования Kotlin с использованием Supabase в роли хранилища данных.

Ресурсы:

* IDE: Android Studio
* Фреймворк: Jetpack Compose
* Паттерн: MVVM
* Язык: Kotlin
* База данных: Supabase
* Документация: Microsoft Word, Markdown

Ответственность в команде:

Разработчик Екатерина ответственна за все аспекты разработки, включая проектирование, программирование функционала, тестирование и документирование.

Распределение ролей представлено в таблице 2.

Таблица 2 – Распределение ролей участников

|  |  |  |
| --- | --- | --- |
| **Стадия** | **Время** | **Участники** |
| **Постановка целей и задач** | | |
| Разработка технического задания | 2 часа | Екатерина |
| Выбор паттерна проектирования | 30 минут | Екатерина |
| Проектирование требований к серверной части приложения | 30 минут | Екатерина |
| Проектирование требований к мобильному приложению | 30 минут | Екатерина |
| Выбор для реализации ПО и обоснование выбора | 30 минут | Екатерина |
| **Разработка макета приложения** | | |
| Разработка макета каждого экрана разрабатываемого мобильного приложения (выбор шрифтов, цветовой палитры) | 6 часов | Екатерина |
| **Разработка серверной части приложения** | | |
| Проектирование серверной части в Supabase | 2 часа | Екатерина |
| Реализация серверной части в мобильном приложении | 2 часа | Екатерина |
| **Тестирование серверной части приложения** | | |
| Разработка документации для тестирования сервера приложения (тест-план, тестовый набор, тест-кейс) | 3 часа | Екатерина |
| Спроектировать и реализовать 10 unit-test к серверу приложения | 1 час | Екатерина |
| **Разработка мобильного приложения** | | |
| Подключение шрифтов и настройка ранее выбранных цветов | 1 час | Екатерина |
| Верстка макета | 15 часов | Екатерина |
| Подключение библиотек из интернета | 1 час | Екатерина |
| Подключение к серверной части приложения | 5 часов | Екатерина |
| Разработка вспомогательных методов, и методов для обработки данных с сервера. | 4 часа | Екатерина |
| **Тестирование мобильного приложения** | | |
| Разработка тестовой документации (тест план, тестовый набор, тест-кейс) | 2 часа | Екатерина |
| Проектирование и реализация 5 unit-test и 5 ui-test к мобильному приложению | 2 часа | Екатерина |

* 1. **Порядок контроля и приёмки системы**
     + 1. Виды и объем испытаний системы

Система подвергается испытаниям следующих видов:

1. Предварительные испытания.

2. Опытная эксплуатация.

3. Приемочные испытания.

Состав, объем и методы предварительных испытаний системы определяются документом «Программа и методика испытаний», разрабатываемым на стадии «Рабочая документация».

Состав, объем и методы опытной эксплуатации системы определяются документом «Программа опытной эксплуатации», разрабатываемым на стадии «Ввод в действие».

Состав, объем и методы приемочных испытаний системы определяются документом «Программа и методика испытаний», разрабатываемым на стадии «Ввод в действие» с учетом результатов проведения предварительных испытаний и опытной эксплуатации.

* + - 1. Требования к приемке работ по стадиям

Участники испытаний: организации Заказчика и Разработчика.

Место проведения: на территории Заказчика.

Таблица 3 – Требования к приемке работ по стадиям

|  |  |  |
| --- | --- | --- |
| Стадия испытаний | Порядок согласования документации | Статус приемочной комиссии |
| Предварительные испытания | Проведение предварительных испытаний.  Фиксирование выявленных неполадок в Протоколе испытаний.  Устранение выявленных неполадок.  Проверка устранения выявленных неполадок.  Принятие решения о возможности передачи приложения в опытную эксплуатацию.  Составление и подписание Акта приемки приложения в опытную эксплуатацию. | Экспертная группа |
| Опытная эксплуатация | Проведение опытной эксплуатации.  Фиксирование выявленных неполадок в Протоколе испытаний.  Устранение выявленных неполадок.  Проверка устранения выявленных неполадок.  Принятие решения о готовности приложения к приемочным испытаниям.  Составление и подписание Акта о завершении опытной эксплуатации приложения. | Группа тестирования |
| Приемочные испытания | Проведение приемочных испытаний.  Фиксирование выявленных неполадок в Протоколе испытаний.  Устранение выявленных неполадок.  Проверка устранения выявленных неполадок.  Принятие решения о возможности передачи приложения в промышленную эксплуатацию.  Составление и подписание Акта о завершении приемочных испытаний и передаче приложения в промышленную эксплуатацию.  Оформление Акта завершения работ. | Приемочная комиссия |

* 1. **Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие**
     + 1. Технические мероприятия

1. Установка и настройка специализированного программного обеспечения Meow Mates.
2. Интеграция новой системы с уже существующими информационными системами, если это необходимо.
3. Обеспечение безопасности данных, включая регулярное резервное копирование информации в Supabase.
   * + 1. Организационные мероприятия
4. Определение ответственных лиц за внедрение, обучение и поддержку системы.
5. Разработка процедур работы с приложением, включая правила добавления питомцев, поиск и фильтрацию котиков.
6. Проведение обучающих семинаров и тренингов для пользователей по работе с приложением.
7. Мониторинг работы приложения, анализ его эффективности и внесение необходимых корректировок.
   * + 1. Дополнительные мероприятия
8. Проведение обучения пользователей по работе с приложением.
9. Планирование и организация регулярного технического обслуживания и поддержки приложения.
   1. **Требования к документированию**

Документирование MM-Simulator должно содержать:

1. Разработка плановой документации, описывающей применение и варианты использования приложения.
2. Разработка тестовой документации, содержащей тест-план, тестовую стратегию, тестовые сценарии и тестовые наборы.
3. Использование документации в репозитории в формате README файла, с ссылкой на более подробную документацию в формате Markdown.
   1. **Источники разработки**

Настоящее Техническое Задание разработано на основе следующих документов и информационных материалов:

* ГОСТ 34.602-89 «Техническое задание на создание автоматизированной системы».
* ГОСТ 12.4.131-83 «Система стандартов безопасности труда. Средства индивидуальной защиты. Общие требования к проверке».
* ГОСТ 12.1.004-91 «ССБТ. Пожарная безопасность. Общие требования».
* ГОСТ 24.701-86 «Надежность автоматизированных систем управления».

## Выбор ПО и паттерна проектирования

Для мобильного приложения «Meow Mates2 был выбран паттерн проектирования MVVM (Model-View-ViewModel). Этот паттерн позволяет эффективно разделять логику приложения и пользовательский интерфейс, что особенно важно для мобильных приложений, где удобство взаимодействия с пользователем имеет первостепенное значение.

Обоснование выбора MVVM:

1. Четкое разделение ответственности:

* Model отвечает за данные и бизнес-логику приложения (например, информация о питомцах, пользователях и их взаимодействиях).
* View представляет пользовательский интерфейс (UI), который отображает данные и реагирует на действия пользователя.
* ViewModel служит связующим звеном между Model и View, обеспечивая привязку данных и управление состоянием UI.

1. Упрощение тестирования: Паттерн MVVM позволяет легко тестировать бизнес-логику, так как ViewModel может быть протестирован независимо от View. Это упрощает процесс написания юнит-тестов.
2. Поддержка реактивного программирования: Использование LiveData и других реактивных компонентов в Android позволяет View автоматически обновляться при изменении данных в ViewModel, что улучшает пользовательский опыт.
3. Удобство разработки: MVVM хорошо интегрируется с библиотеками Android Jetpack, такими как LiveData и ViewModel, что упрощает разработку и уменьшает количество кода.

## ВЫБОР ДЛЯ РЕАЛИЗАЦИИ ПО

Для реализации мобильного приложения «Meow Mates» были выбраны следующие технологии:

**3.1. Язык программирования: Kotlin**

Обоснование выбора:

* Kotlin является официальным языком для разработки под Android, что обеспечивает хорошую поддержку со стороны Google.
* Он предлагает лаконичный синтаксис и современные возможности, такие как расширения функций и корутины для асинхронного программирования.
* Kotlin совместим с Java, что позволяет использовать существующие библиотеки и фреймворки.

**3.2. Среда разработки: Android Studio**

Обоснование выбора:

* Android Studio предоставляет мощные инструменты для разработки, отладки и тестирования приложений.
* Она включает в себя встроенные средства для работы с Gradle, что упрощает управление зависимостями и сборкой проекта.
* Поддерживает интеграцию с Git для управления версиями кода.

**3.3. Хранилище данных: Supabase**

Обоснование выбора:

* Supabase предоставляет облачное решение для хранения данных с поддержкой PostgreSQL, что обеспечивает надежность и масштабируемость.
* Он предлагает простую интеграцию с приложениями на Kotlin через RESTful API или библиотеку клиентского SDK.
* Supabase включает функции аутентификации и управления пользователями, что упрощает реализацию безопасности в приложении.

**3.4. UI-фреймворк: Jetpack Compose**

Обоснование выбора:

* Jetpack Compose позволяет создавать пользовательские интерфейсы декларативным способом, что упрощает разработку и поддержку UI.
* Он интегрируется с другими компонентами Jetpack, такими как LiveData и ViewModel, что делает его идеальным выбором для использования с паттерном MVVM.
* Compose обеспечивает высокую производительность и возможность кастомизации интерфейса без необходимости написания большого объема кода.

# ГЛАВА 2. РАЗРАБОТКА МАКЕТА

## Макет каждого экрана разрабатываемого приложения.

Макет был выполнен в Figma и представлен по ссылке: <https://www.figma.com/design/w2HXW4Iq7LtwhHI8sro678/Meow-mates?node-id=0-1&t=nz9VY0NWUA284gup-1> .

## Выбор цветовой палитры

Разработаны две цветовые темы: тёмная (Рис. 1 ниже) и светлая (Рис. 2 ниже).

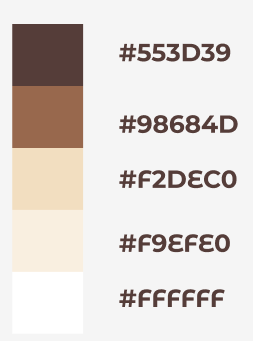


Рисунок 1. Цветовая схема светлой т емы приложения

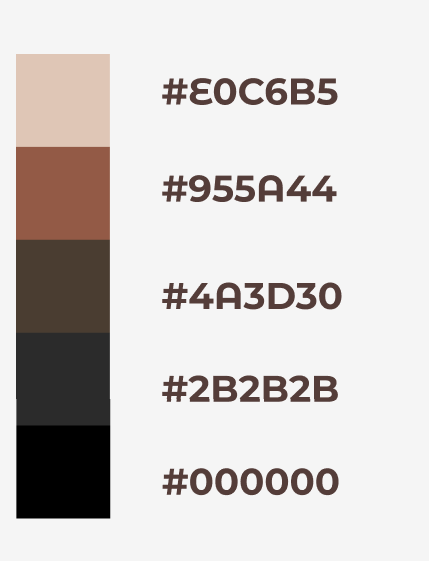


Рисунок 2. Цветовая схема темной темы приложения

## Выбор шрифта для заголовков и основного текста

Шрифт, используемый в макете приложения указан на Рисунке 3 ниже.



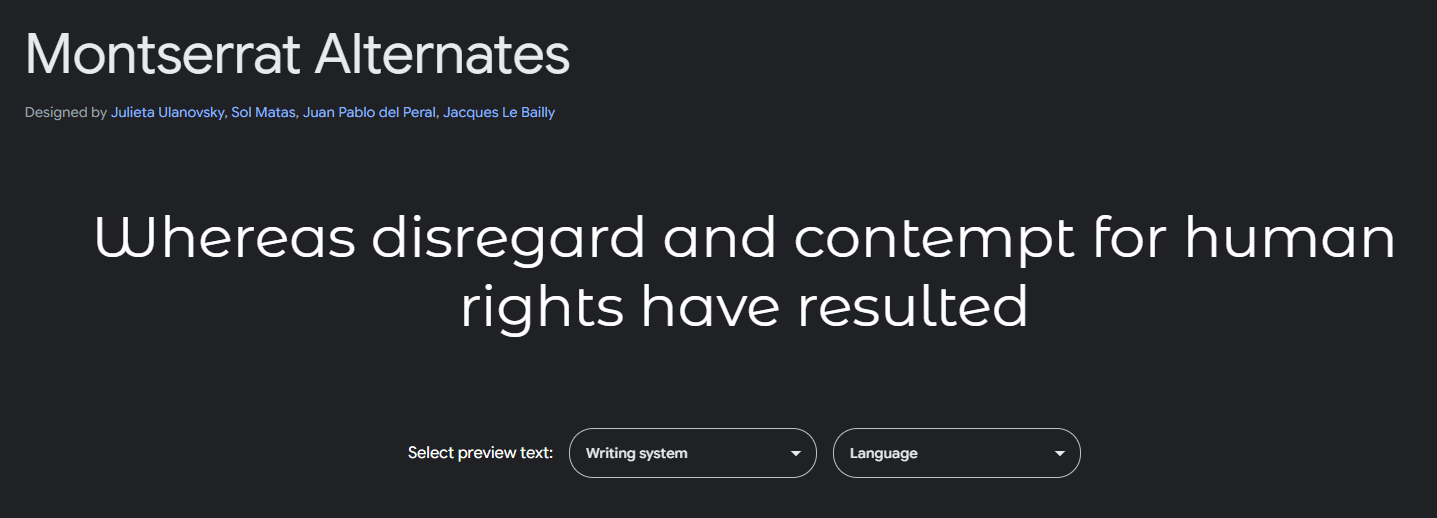


Рисунок 3. Шрифты приложения.

## Элементы формы

В на отжельном экране были вынесены все элементы приложения (иконки, картинки по умолчанию). Их можно увидеть на Рис. 5 ниже



Рисунок 5. Элементы формы.

## Разработка логотипа

Как такогого логитипа у приложения нет, но приложение активно использует название Meow Mates, написанное специальным шрифтом Cat Regular. Выглядит следующим образом:



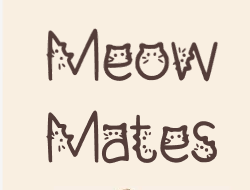


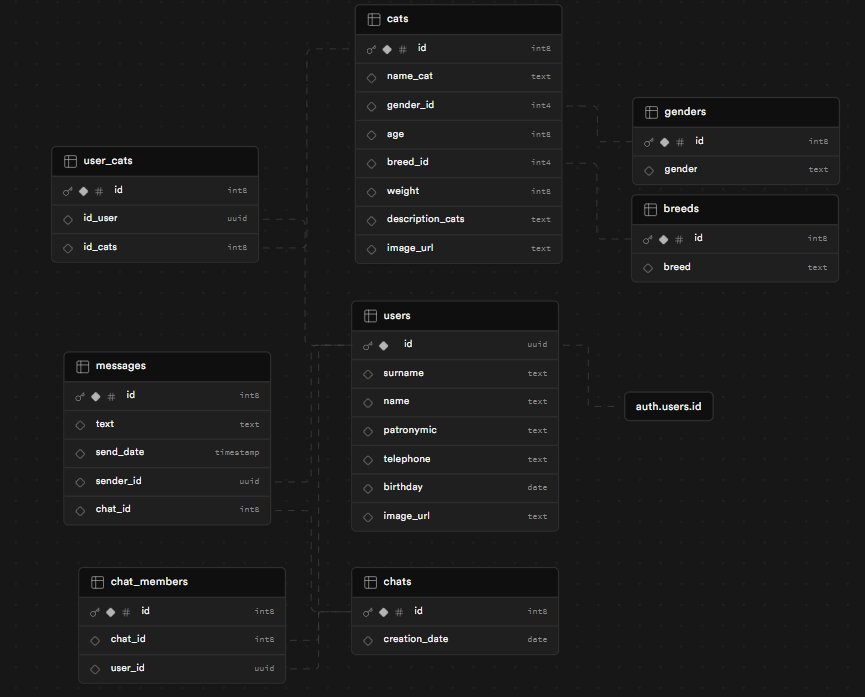
Рисунок 6. Логотип.

## Инструменты Figma.

Если перейти по [ссылке макета](https://www.figma.com/design/w2HXW4Iq7LtwhHI8sro678/Meow-mates?node-id=0-1&t=nz9VY0NWUA284gup-1), откроется главная страница макета с описанием стилей, того, как выглядит приложение в нескольких цветовых темах (светлая, темная). Описание того, какие шрифты необходимо использовать.

# ГЛАВА 3. РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ

Была разработана следующая схема базы данных:



А теперь, о каждой таблице поподробней:

* 1. **Таблица breeds:**

Данная таблица хранит информацию о породе кота (кот/кошка) и имеет следующие поля:

|  |  |  |
| --- | --- | --- |
| PK | Поле | Тип данных |
| PK | id | Int8 |
|  | breed | text |

* 1. **Таблица genders:**   
     Данная таблица хранит информацию о гендере кота (кот/кошка) и имеет следующие поля:

|  |  |  |
| --- | --- | --- |
| PK | Поле | Тип данных |
| PK | id | Int8 |
|  | gender | text |

* 1. **Таблица cats:**

Данная таблица хранит информацию о котах (имя, порода, возраст, вес, описание, картинку, пол кота, код породы) и имеет следующие поля:

|  |  |  |
| --- | --- | --- |
| PK | Поле | Тип данных |
| PK | id | Int8 |
|  | name\_cat | text |
| FK | gender\_id | Int8 |
|  | age | Int8 |
| FK | breed\_id | Int8 |
|  | wheight | Int8 |
|  | description\_cats | text |
|  | Image\_url | text |

Поле gender\_id ссылается на таблицу genders на поле id. Поле breed\_id ссылается на таблицу breeds на поле id.

* 1. **Таблица users:**

Данная таблица хранит информацию о хозяине кота (ФИО, телефон, дату рождения, ссылку на картинку) и имеет следующие поля:

|  |  |  |
| --- | --- | --- |
| PK | Поле | Тип данных |
| PK | id | uuid |
|  | surname | text |
|  | name | text |
|  | patronymic | text |
|  | telephone | text |
|  | birthday | text |
|  | Image\_url | text |

Поле id ссылается на встроенную схему авторизации в supabase (auth), на таблицу users на поле id. Благодаря этому соединению мы можем применять интегрированный механизм аутентификации от Supabase и одновременно сохранять дополнительные сведения о залогиненном пользователе.

* 1. **Таблица user\_cats:**

Данная таблица позволяет связать 2 таблицы связью многое-ко-многим. Она имеет следующие поля:

|  |  |  |
| --- | --- | --- |
| PK | Поле | Тип данных |
| PK | id | Int8 |
| FK | Id\_user | uuid |
| FK | Id\_cats | Int8 |

Поле id\_user ссылается на таблицу users на поле id. Поле id\_cats ссылается на таблицу cats на поле id. Эти отношения представляют собой связь «один ко многим» между таблицами пользователей и котов. У одного владельца может быть несколько питомцев, но один питомец не может принадлежать нескольким владельцам.

* 1. **Таблица chats:**

Данная таблица хранит информацию о чатах (номере и дате создания) имеет следующие поля:

|  |  |  |
| --- | --- | --- |
| PK | Поле | Тип данных |
| PK | id | Int8 |
|  | Creation\_date | date |

* 1. **Таблица chat\_members:**

Данная таблица имеет следующие поля:

|  |  |  |
| --- | --- | --- |
| PK | Поле | Тип данных |
| PK | id | Int8 |
|  | Chat\_id | Int8 |
|  | user\_id | Int8 |

Поле user\_id ссылается на таблицу users на поле id. Поле chat\_id ссылается на таблицу chats на поле id. Эти всвязи создают отношения «многие ко многим» между таблицами пользователей и чатов. В одном чате может быть несколько участников, как минимум двое, а один и тот же человек может быть участником нескольких чатов.

* 1. **Таблица messages:**

Данная таблица хранит информацию о сообщениях (сообщение, дату отправки, кто отправил, в каком чате) и имеет следующие поля:

|  |  |  |
| --- | --- | --- |
| PK | Поле | Тип данных |
| PK | id | uuid |
|  | text | text |
|  | Send\_date | timestamp |
|  | Sender\_id | uuid |
|  | Chat\_id | Int8 |

# ГЛАВА 4. ТЕСТИРОВАНИЕ СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ

Документация тестирования серверной части представлена в [Приложении 3](#_Приложение_3).

# ГЛАВА 5. РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ

## Используемые библиотеки

1. Библиотеки для работы с JWT (JSON Web Tokens) — для декодирования и верификации токенов.

|  |
| --- |
| *implementation* ("io.jsonwebtoken:jjwt-api:0.12.6") *runtimeOnly* ("io.jsonwebtoken:jjwt-impl:0.12.6") *implementation* ("com.auth0.android:jwtdecode:2.0.2") *runtimeOnly* ("io.jsonwebtoken:jjwt-gson:0.12.6") |

1. Библиотека Jetpack Compose для реализации навигации между экранами в вашем приложении.

|  |
| --- |
| val nav\_version = "2.7.7" *implementation*("androidx.navigation:navigation-compose:$nav\_version") |

1. Набор библиотек Ktor для создания HTTP-клиента. Используются различные модули для базовой функциональности, работы с Android, логирования, обработки контента и сериализации/десериализации JSON с помощью kotlinx.serialization.

|  |
| --- |
| val ktor\_version = "2.3.10" *implementation*("io.ktor:ktor-client-core:$ktor\_version") *implementation*("io.ktor:ktor-client-android:$ktor\_version") *implementation*("io.ktor:ktor-client-logging:$ktor\_version") *implementation*("io.ktor:ktor-client-content-negotiation:$ktor\_version") *implementation*("io.ktor:ktor-serialization-kotlinx-json:$ktor\_version") |

1. Библиотека для логирования, используемая в сочетании с Ktor для вывода логов.

|  |
| --- |
| *implementation*("ch.qos.logback:logback-classic:1.2.3") |

1. Библиотека Compose для реализации компонента “Pull to refresh” (обновление при прокрутке вниз).

|  |
| --- |
| *implementation*("eu.bambooapps:compose-material3-pullrefresh:1.0.0") |

1. Библиотеки Hilt (DI фреймворк) для внедрения зависимостей.

|  |
| --- |
| val hilt\_android\_version = "2.48" *implementation*("com.google.dagger:hilt-android:$hilt\_android\_version") *kapt*("com.google.dagger:hilt-compiler:$hilt\_android\_version") *implementation*("androidx.hilt:hilt-navigation-compose:1.0.0") |

1. Библиотека для написания UI-тестов для Jetpack Compose с использованием JUnit 4.

|  |
| --- |
| val compose\_version = "1.7.5" *androidTestImplementation*("androidx.compose.ui:ui-test-junit4:$compose\_version") |

1. Библиотеки Room для работы с локальной базой данных.

|  |
| --- |
| val room\_version = "2.4.0" *implementation*("androidx.room:room-runtime:$room\_version") *kapt*("androidx.room:room-compiler:$room\_version") *implementation* ("androidx.room:room-ktx:$room\_version") |

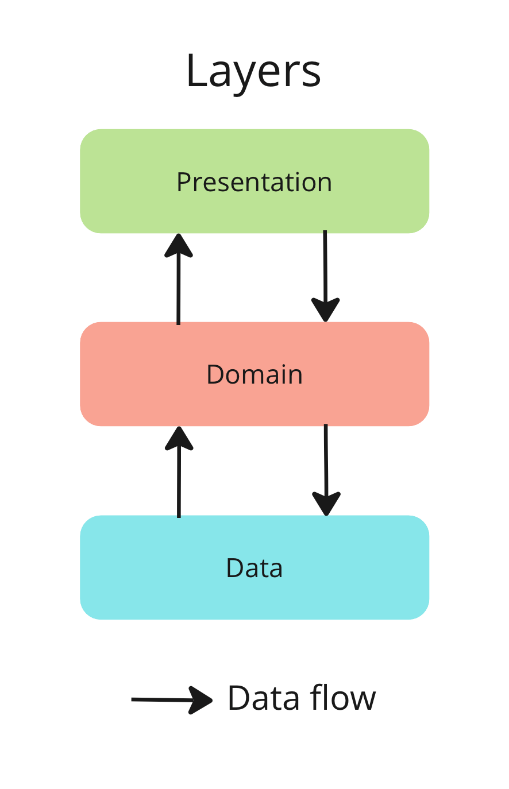
## Архитектура

Выбранная архитектура - Clean Architecture. Такой выбор был сделан исходя из следующих плюсов:

1. Разделение ответственности (масштабируемость, минимизация проблем с зависимостями)
2. Тестируемость
3. Устойчивость к изменениям
4. Популярность (большое кол-во гайдлайнов, шаблонов)

Принципы:

1. Единственная ответственность. Каждый слой (модуль), класс или функция выполняет только одну задачу. Данный принцип призван конкретизировать ответственность отдельных по смыслу классов или слоев, что в свою очередь придает ясность в код и снижает его связанность.
2. Разделение на слои. Приложение должно быть разбито на слои, у каждого слоя своя зона ответственности. Обычно выделяют следующие уровни: presentation, domain, data.
   1. Presentation: отвечает за отображение пользовательского интерфейса и реагирование на его события
   2. Domain: бизнес-логика, изолированная от деталей реализации, определяет правила и операции, как приложение должно взаимодействовать с данными
   3. Data: хранилище данных



1. Инверсия зависимостей. Один из важнейших принципов который гласит о том, что стоит использовать общий контракт, такой как интерфейс или абстрактный класс, вместо прямой зависимости слоя верхнего уровня от компонентов слоя нижнего. Таким образом, каждый слой использует этот контракт, что обеспечивает изоляцию изменений в верхнем слое.

Архитектура проекта представлена на Рисунке 7 ниже.

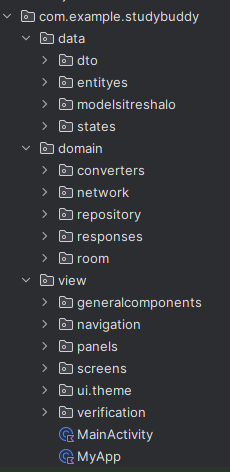


Рисунок 7. Архитектура проекта.

## Тема

После создания приложения на Jetpack Compose создаётся базовая тема Material Theme. Но данную тему не очень удобно использовать, поэтому была создана новая, custom тема StudyBuddyTheme.

|  |
| --- |
| @Composable fun StudyBuddyTheme(  themeMode: ThemeMode = ThemeMode.Dark,  typography: Typography = StudyBuddyTheme.typography,  content: @Composable () -> Unit ) {  val colors = when (themeMode) {  ThemeMode.Dark -> *darkPalette* ThemeMode.Light -> *baseLightPalette* }  CompositionLocalProvider(  *LocalTypography* provides typography,  *LocalColors* provides colors  )**{** content()  **}** } |

В эту тему можно передать аргументом цветовую режим темы (светлая, тёмная).

Также для удобства был создан класс темы, чтобы задавать свои названия переменным с цветом.

|  |
| --- |
| data class ColorPalette(  val background: Color,  val primary: Color,  val secondary: Color,  val third: Color,  val textTitle: Color,  val textDesc: Color,  val textButton: Color,  val containerPrimary: Color,  val containerSecondary: Color,  val containerThird: Color,  val containerDefault: Color,  val unselectItem: Color ) |

На основе этого класса были добавлены две темы: темная и светлая.

|  |
| --- |
| val *Cream* = *Color*(0xFFFDFAF4) val *DarkBack* = *Color*(0xFF2F2C2B) val *DarkCont* = *Color*(0xFF242424) val *White* = *Color*(0xFFFFFFFF) val *White40* = *Color*(0x66FFFFFF) val *Orange* = *Color*(0xFFEE9D5B) val *Orange2* = *Color*(0xFFFA9442) val *Gray* = *Color*(0xFFB4B8C0) val *Dark\_Green* = *Color*(0xFF4D544C) val *Green* = *Color*(0xFF899C6F) val *Green2* = *Color*(0xFF89A95E) val *Yellow* = *Color*(0xFFF8DA7A)  val *baseLightPalette* = ColorPalette(  background = *Cream*,  primary = *Green*,  secondary = *Orange*,  third = *Yellow*,  textTitle = *Dark\_Green*,  textDesc = *Green*,  textButton = *White*,  containerPrimary = *Green*.copy(alpha = 0.2f),  containerSecondary = *Orange*.copy(alpha = 0.2f),  containerThird = *Yellow*.copy(alpha = 0.8f),  containerDefault = *White*,  unselectItem = *Gray* )  val *darkPalette* = ColorPalette(  background = *DarkBack*,  primary = *Green2*,  secondary = *Orange2*,  third = *Yellow*,  textTitle = *White*,  textDesc = *Green2*,  textButton = *White*,  containerPrimary = *Green2*.copy(alpha = 0.2f),  containerSecondary = *Orange2*.copy(alpha = 0.2f),  containerDefault = *DarkCont*,  containerThird = *Yellow*.copy(alpha = 0.8f),  unselectItem = *White40* )  val *LocalColors* = *staticCompositionLocalOf* **{** *darkPalette* **}** |

Точно таким же способом создан класс для шрифтов, в котором добавлены шрифты и стили для текста

|  |
| --- |
| data class Typography(  val bold: TextStyle,  val regular: TextStyle,  val extralight: TextStyle,  val light: TextStyle )  val *Geologica* = *FontFamily*(  *Font*(R.font.*geologica\_black*, FontWeight.Black),  *Font*(R.font.*geologica\_bold*, FontWeight.Bold),  *Font*(R.font.*geologica\_extra\_bold*, FontWeight.ExtraBold),  *Font*(R.font.*geologica\_extra\_light*, FontWeight.ExtraLight),  *Font*(R.font.*geologica\_light*, FontWeight.Light),  *Font*(R.font.*geologica\_medium*, FontWeight.Medium),  *Font*(R.font.*geologica\_regular*, FontWeight.Normal),  *Font*(R.font.*geologica\_semi\_bold*, FontWeight.SemiBold),  *Font*(R.font.*geologica\_thin*, FontWeight.Thin) )  val *typography* = Typography(  bold = TextStyle(  fontWeight = FontWeight.Bold,  fontFamily = *Geologica*,  fontSize = 16.*sp* ),  regular = TextStyle(  fontWeight = FontWeight.Normal,  fontFamily = *Geologica* ),  extralight = TextStyle(  fontWeight = FontWeight.ExtraLight,  fontFamily = *Geologica* ),  light = TextStyle(  fontWeight = FontWeight.Light,  fontFamily = *Geologica* ) )  val *LocalTypography* = *staticCompositionLocalOf* **{** *typography* **}** |

В MainActicity установлена тема

|  |
| --- |
| val currentThemeMode = remember **{** *mutableStateOf*(UserRepository.themes.*first* **{ it**.title == UserRepository.theme **}**) **}** StudyBuddyTheme(themeMode = currentThemeMode.value) **{** Scaffold() **{** Navigation()  **} }** |

## Настройка DI (Hilt).

В процессе разработки мобильного приложения, многие классы (например, viewModel) имеют зависимости, которые необходимо передавать при создании объекта. Зависимостей может быть достаточно много и для того, чтобы каждый раз для создания объекта с зависимостями не нужно было прописывать и сами зависимости будет использована библиотека Hilt, которая будет автоматически генерировать нужные зависимости (для этого нужно только прописать модуль для генерации)

Все приложения, использующие Hilt, должны содержать класс Application, помеченный @HiltAndroidApp.

|  |
| --- |
| @HiltAndroidApp class MyApp : Application() {  } |

@HiltAndroidApp запускает генерацию кода Hilt, включая базовый класс для приложения, который служит контейнером зависимостей на уровне приложения.

Этот сгенерированный компонент Hilt прикрепляется к жизненному циклу объекта Application и предоставляет ему зависимости. Кроме того, это родительский компонент приложения, а это означает, что другие компоненты могут получить доступ к предоставляемым им зависимостям.

Также необходимо указать его в Manifest в атрибуте name

|  |
| --- |
| <application  android:name=".view.MyApp"  /// </application> |

Также, чтобы Hilt предоставлял зависимости другим классам Android, у MainActivity прописана аннотация @AndroidEntryPoint

|  |
| --- |
| @AndroidEntryPoint class MainActivity : ComponentActivity() {  ///  } |

## Локальная база данных Room

Для организации хэширования данных используется библиотека Room.

Room — это ORM (Object-Relational Mapping) библиотека от Android, упрощающая работу с базами данных SQLite. Он работает по принципу абстрагирования доступа к базе данных, предоставляя более удобный объектно-ориентированный интерфейс.

1. **Entity:** это классы, которые представляют таблицы в базе данных. Они аннотируются (@Entity), и поля класса соответствуют столбцам таблицы.
2. **DAO (Data Access Object):** это интерфейсы (или абстрактные классы), которые определяют методы доступа к базе данных. Эти методы аннотируются (@Insert, @Update, @Delete, @Query), и Room генерирует реализацию этих методов, которые выполняют запросы к базе данных.
3. **Database:** Это абстрактный класс (@Database), который объявляет все Entity, используемые в приложении, и содержит метод getDao(), возвращающий экземпляр DAO. Room генерирует реализацию этого класса, которая обрабатывает создание и управление базой данных.

В essence, вы определяете структуру данных (Entity), методы доступа к ним (DAO) и объединяете все это в единый объект базы данных (@Database). Room берет на себя всю работу по преобразованию объектно-ориентированного кода в SQL-запросы и обратно, что значительно упрощает разработку и повышает безопасность.

Для генерации аргумента типа RoomDatabase() была прописана в классе с аннотацией @Module функция, возвращающая объект типа StudyBuddyDatabase.

|  |
| --- |
| @Provides fun provideTaskDb(  @ApplicationContext  context: Context ) = Room.databaseBuilder(  context,  StudyBuddyDatabase::class.*java*,  "tasks" ).allowMainThreadQueries().build() |

## Подключение к API

Для подключения к API используется Ktor Client.

Создан модуль, в котором описана инструкция (функция), как генерировать аргумент типа HttpClient. В этой функции идёт настройка KtorClient.

|  |
| --- |
| @Module @InstallIn(SingletonComponent::class) class ApiServiceProvider {   @Provides  fun provideClient(): HttpClient {  return HttpClient(Android)**{** expectSuccess = true  install(Logging) **{** level = LogLevel.*ALL* **}** install(ContentNegotiation)**{** *json*(  *Json* **{** encodeDefaults = false  ignoreUnknownKeys = true  isLenient = true  useAlternativeNames = false  **}**)  register(  ContentType.Text.Html, KotlinxSerializationConverter(  *Json* **{** prettyPrint = true  isLenient = true  ignoreUnknownKeys = true  **}** )  )  **}  }** }  ///  } |

Был создан интерфейс, в котором описаны будущие методы для обращения к API и на его основе был реализован класс, в котором реализованы все методы подключения к API.

|  |
| --- |
| */\*\* Реализация интерфейса, в котором описаны все методы для запросов к API + их кэшировние в локальную базу данных Room\*/* class ApiServiceImpl(  private val client: HttpClient,  private val database: StudyBuddyDatabase,  ): ApiService {  ///Методы подключения к API  } |

Объект данного класса используется во viewModel, поэтому необходимо прописать провайдера в модуль Hilt.

|  |
| --- |
| @Provides fun provideService(client: HttpClient, database: StudyBuddyDatabase): ApiServiceImpl {  return ApiServiceImpl(client, database) } |

И теперь с помощью Hilt во ViewModel легко внедрять зависимости для подключения к API и для обращения к локальной БД, также легко передавать контекст приложения с помощью аннотации @ApplicationContext.

|  |
| --- |
| @HiltViewModel class ExamsViewModel @Inject constructor(  private val service: ApiServiceImpl,  @ApplicationContext private val context: Context,  private val database: StudyBuddyDatabase, ) : ViewModel() {  ///  } |

## Функции валидации и регулярные выражения

## Функции

ГЛАВА 6. ТЕСТИРОВАНИЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

ЗАКЛЮЧЕНИЕ

Выводы:

Рекомендации:

# Приложение 1

# Приложение 2

Выбор ПО и паттерна проектирования

1. **ВЫБОР ПО ДЛЯ РЕАЛИЗАЦИИ.**
   1. **Мобильное приложение.**

Android Studio будет использоваться в качестве среды разработки для мобильного приложения. Android Studio – это официальная среда разработки от OpenAI для Android, которая предлагает множество преимуществ для разработки приложений:

* Интеграция с Android SDK: Android Studio тесно интегрирована с Android SDK, что значительно упрощает процесс разработки и отладки.
* Удобные инструменты: Android Studio предоставляет богатый набор инструментов, таких как визуальный редактор макетов, инструменты профилирования и отладки, а также эмулятор Android.
* Система сборки Gradle: Android Studio использует систему сборки Gradle, которая обеспечивает гибкость и контроль над процессом сборки приложения.
* Инструменты для тестирования: Android Studio предоставляет инструменты для автоматического тестирования (Android Test и Espresso), что позволяет разработчикам убедиться в качестве кода.
* Система управления версиями: Android Studio интегрирована с системами управления версиями, такими как Git, что облегчает совместную разработку и отслеживание изменений.

Для написания кода приложения будет использоваться Jetpack Compose с использованием языка Kotlin. Такой выбор был сделан по ряду причин:

* Современный декларативный UI-фреймворк: Jetpack Compose – это современный UI-фреймворк от OpenAI, который предлагает декларативный подход к созданию пользовательского интерфейса, что делает код более лаконичным и читаемым.
* Упрощение UI: Jetpack Compose упрощает разработку UI, так как позволяет описывать UI в виде композиции функций.
* Интеграция с другими компонентами Jetpack: Jetpack Compose хорошо интегрируется с другими компонентами Jetpack, такими как ViewModel и LiveData, что делает разработку более структурированной.
* Поддержка Material Design: Jetpack Compose включает в себя поддержку Material Design, что позволяет создавать приложения с современным и привлекательным интерфейсом.
* Простота изучения: Jetpack Compose относительно прост в изучении, что сокращает время разработки и позволяет быстро приступить к работе.
* Минимальный код: Jetpack Compose позволяет создавать UI с меньшим количеством кода по сравнению с традиционными методами разработки UI на Android.
  1. **Серверная часть.**

Для реализации серверной части необходима база данных и API, с помощью которого можно будет управлять данными базы при помощи телефона.

Данные будут храниться на сервере НГК в СУБД Postgre SQL, а API будет разработана с помощью фреймворка ASP.NET.

Такой выбор был сделан из-за ряда преимуществ:

* Надежность и стабильность: PostgreSQL известна своей высокой стабильностью и надежностью, что делает ее отличным выбором для приложений.
* Транзакционная целостность: PostgreSQL поддерживает ACID-свойства (атомарность, согласованность, изоляция, долговечность), гарантируя целостность данных при одновременном доступе.
* Широкие возможности моделирования данных: PostgreSQL поддерживает сложные типы данных, в том числе JSON, геопространственные данные, массивы и пользовательские типы.
* Высокая производительность: PostgreSQL обеспечивает высокую производительность и масштабируемость, оптимизирована для работы с большими объемами данных.
* Открытый исходный код: PostgreSQL бесплатна для использования и распространения, что делает ее доступной для разработчиков всех уровней.
* Гибкость: ASP.NET Core предоставляет широкие возможности для кастомизации, чтобы создать API, идеально соответствующее требованиям.
* Масштабируемость: возможность легко масштабировать приложение, добавляя серверы и ресурсы по мере роста нагрузки.
* Опыт: ASP.NET - устоявшийся фреймворк с большим сообществом и множеством ресурсов.
* Открытый исходный код: PostgreSQL и ASP.NET Core бесплатны и открыты, что дает свободу использования и модификации.

1. **ВЫБОР ПАТТЕРНА ПРОЕКТИРОВАНИЯ.**

Выбранный паттерн проектирования: MVVM (Model-View-ViewModel)

MVVM (Model-View-ViewModel) – это архитектурный паттерн, который широко применяется в разработке мобильных приложений. Он разделяет приложение на три основных компонента:

1. Модель (Model): представляет данные приложения, логику доступа к данным (например, работа с базой данных) и правила валидации данных.
2. Представление (View): отображает данные и взаимодействует с пользователем. Это то, что видит пользователь (например, экраны, элементы UI).
3. Модель представления (ViewModel): является посредником между моделью и представлением. Он обрабатывает запросы от представления, получает данные от модели, форматирует их и передает обратно в представление. ViewModel не должен иметь зависимостей от View, поэтому его можно использовать для тестирования и повторного использования.

Почему выбор пал на MVVM для разработки мобильного приложения:

* Разделение ответственности: MVVM помогает разделить ответственность между разными компонентами приложения, что делает код более структурированным и легко поддерживаемым.
* Тестируемость: MVVM делает приложение более тестируемым, так как ViewModel не зависит от View и может быть тестирован отдельно.
* Повторное использование: ViewModel можно повторно использовать в разных частях приложения или в других приложениях.
* Обновление UI: MVVM делает обновление UI более простым, так как ViewModel уведомляет View об изменениях данных с помощью механизмов наблюдения.
* Совместимость с Jetpack Compose: MVVM хорошо интегрируется с Jetpack Compose, позволяя создавать более структурированные и тестируемые UI.

# ПРИЛОЖЕНИЕ А

Документация тестирования серверной части приложения

# 1. Тест план.

# Введение.

Данный тест-план описывает стратегию тестирования базы данных приложения Meow Mates, которая управляет информацией о котах, их владельцах и взаимодействиями между пользователями через чаты. Тестирование будет проводиться с использованием Supabase как платформы для управления базой данных.

# Цели тестирования.

Цель: Обеспечить корректность, производимость и безопасность базы данных.

1. проверка корректности работы CRUD операций (создание, чтение, обновление, удаление) для всех таблиц.
2. убедиться в правильности работы внешних ключей и связей между таблицами;
3. проверка правильности применения политик безопасности (RLS);
4. проверка целостности данных.

# Область тестирования

В базе данных есть следующие таблицы:

* breeds: информация о породах котов.
* genders: информация о гендерах котов.
* cats: информация о котах (имя, порода, возраст и т.д.).
* users: информация о владельцах котов (ФИО, телефон и т.д.).
* user\_cats: связь между пользователями и их котами.
* chats: информация о чатах.
* chat\_members: участники чатов.
* messages: сообщения в чатах

# Стратегия тестирования

Подход к тестированию: Автоматизированное. Создание Unit-тестов для автоматизации проверки функциональности базы данных.

Уровень тестирования: Модульное. Тестирование работы базы данных, без ее взаимодействия с бизнес-логикой будущего мобильного приложения.

# Типы тестов

* Функциональное тестирование: Проверка работы всех функций приложения (регистрация, добавление котов, поиск партнеров и т.д.).
* Интеграционное тестирование: Проверка взаимодействия между различными компонентами приложения (например, связь между пользователями и их котами).

# Ресурсы.

* MSTest- Необходимо для создания Unit-тестов;
* С#: Язык программирования для написания Unit-тестов
* Microsoft Visual Studio – IDE, где будут писаться тесты
* Supabase - Облачное СУБД для хранения данных мобильного приложения, объект тестирования

# Критерии завершения

Тестирование будет завершено, когда:

* все тесты будут успешно пройдены;
* будет подготовлена отчетность о результатах тестирования.

# Тестовый набор

Тестовый набор – это набор тест-кейсов, в которых результат описывается предисловием, то есть очередность проводимых тестов.

* 1. **Выбор фреймворка для тестирования**

Для тестирования API будет использоваться инструмент для модульного тестирования xUnit. xUnit — это пакет с открытым исходным кодом для .NET Framework и .NET Core. Кроме фреймворков для создания и проведения юнит-тестов при тестировании часто бывают полезны такие фреймворки, которые позволяют имитировать или эмулировать какую-то функциональность или создавать мок-объекты. Подобных фреймворков существует множество, и одним из самых популярных является Moq, он и будет использоваться. Также будет использоваться модель тестов Arrange-Act-Assert, которая представляет целую парадигму тестирования, которая используется многими фреймворками юнит-тестов.

* 1. **Проверка аутентификации и авторизации при помощи модульного тестирования.**

В API присутствует система авторизации, поэтому необходимо проверить, корректно ли она работает. Проверка реализована в виде unit тестов, где для изоляции компонентов друг от друга и имитации их взаимодействия используются моки (тестовые объекты, которые имитируют зависимости), также создан тестовый контекст БД. Тестированию подвержен контроллер Account, который отвечает за обработку операций, связанных с учетными записями пользователей, таких как: регистрация, вход в систему, получение информации об авторизованном пользователе. В Таблице 1 ниже указаны тест-кейсы.

Таблица 1. Тест-кейсы для unit-тестов системы авторизации.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Описание** | **Входные данные** | **Ожидаемый результат** | **Проверка** | **Прошел** |
| 1 | Проверка успешной регистрации нового пользователя с валидными данными | 1. Имя пользователя: Mark.  2. Почта: [markmarkovich@mail.ru](mailto:markmarkovich@mail.ru)  3. Пароль: 12345678 | – Возвращается код HTTP 200 (Ok).  – Создается новый пользователь в базе данных и возвращается. | result.Should().NotBeNull();  Assert.NotNull(newUserInContext); result.Should().BeAssignableTo<OkObjectResult>(); | + |
| 2 | Проверка, что при попытке регистрации с уже существующим именем пользователя возвращается ошибка. | 1. Имя пользователя: Mark.  2. Почта: [markmarkovich1@mail.ru](mailto:markmarkovich1@mail.ru)  3. Пароль: 12345678 | – Возвращается код HTTP 400 (BadRequest). | result.Should().NotBeNull();  result.Should().BeAssignableTo<BadRequestObjectResult>(); | + |
| 3 | Проверка успешной аутентификации пользователя с правильными логином и паролем. | 1. Почта: [markmarkovich1@mail.ru](mailto:markmarkovich1@mail.ru)  2. Пароль: 12345678 | - Возвращается код HTTP 200 (Ok). - Генерируется и возвращается JWT-токен. | result.Should().NotBeNull();  Assert.NotNull(resultDto.Token);  result.Should().BeAssignableTo<OkObjectResult>(); | + |
| 4 | Проверка, что при попытке авторизоваться с неправильными логином/паролем возвращается ошибка. | - Имя пользователя: invaliduser - Пароль: invalidpassword | - Возвращается код HTTP 401 (Unauthorized). | result.Should().NotBeNull();  result.Should().BeAssignableTo<UnauthorizedObjectResult>(); | + |
| 5 | Проверка получения информации об авторизованном пользователе | Эмуляция авторизации в системе (через моки) | - Возвращается код HTTP 200 (Ok).  - Возвращается модель данных об аккаунте | result.Should().NotBeNull();  Assert.NotNull(resultDto);  result.Should().BeAssignableTo<ActionResult<UserInfoDto>>(); | + |

* 1. **Проверка работы с данными у авторизованного пользователя.**

После авторизации в системе пользователю доступен ряд GET, POST, DELETE и PUT методов для управления информацией под своей учетной записью:

GET-запросы: используются для получения данных, например, списка экзаменов, информации о профиле пользователя, получение списка требований к предмету и др. Тесты должны проверять, что авторизованный пользователь получает только информацию, относящуюся к его аккаунту.

POST-запросы: используются для создания новых данных, например, добавления нового экзамена. Тесты должны проверять, что данные создаются корректно и привязываются к аккаунту пользователя.

DELETE-запросы: используются для удаления данных, например, удаления экзамена. Тесты должны проверять, что данные действительно удаляются.

PUT-запросы: используются для обновления данных, например, изменения информации об экзамене. Тесты должны проверять, что данные обновляются корректно.

Для проверки были написаны тест-кейсы, они указаны в Таблице 2.

Таблица 2. Тест-кейсы для unit-тестов работы с данными пользователя.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Описание** | **Входные данные** | **Ожидаемый результат** | **Проверка** | **Прошел** |
| 1 | GET - Получение списка экзаменов пользователя | 1. Эмуляция авторизации в системе (через моки). | – Возвращается код HTTP 200 (Ok).  – Список экзаменов у пользователя | result.Should().NotBeNull();  Assert.NotNull(resultDto);  result.Should().BeAssignableTo<ActionResult<List<Exam>>>(); | + |
| 2 | POST – создание экзамена у пользователя | 1. Эмуляция авторизации в системе (через моки).  2. Объект создаваемого экзамена | – Возвращается код HTTP 200 (Ok).  – Добавленный экзамен у пользователя  – Количество экзаменов пользователя меняется в большую сторону | Assert.NotNull(resultDto);  Assert.True(countExamsBefore < countExamsAfter);  result.Should().NotBeNull();  result.Should().BeAssignableTo<ActionResult<Exam>>(); | + |
| 3 | POST – создание экзамена дубликата у пользователя | 1. Эмуляция авторизации в системе (через моки).  2. Объект создаваемого экзамена, который уже есть в БД | - Возвращается код HTTP 400 (BadRequest). | Assert.True(result.Result is BadRequestObjectResult); | + |
| 4 | DELETE –удаление экзамена у пользователя | 1. Эмуляция авторизации в системе (через моки).  2. Id экзамена, который будет удален | – Возвращается код HTTP 200 (Ok). | result.Should().NotBeNull();  result.Should().BeAssignableTo<ActionResult>(); | + |
| 5 | PUT – изменение экзамена у пользователя | 1. Эмуляция авторизации в системе (через моки).  2. Объект экзамена, который будет изменён | - Возвращается код HTTP 200 (Ok). | result.Should().NotBeNull();  result.Should().BeAssignableTo<ActionResult>(); | + |