**Министерство цифрового развития, связи и массовых коммуникаций РФ**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Сибирский государственный университет телекоммуникаций и информатики»**

**(СибГУТИ)**

Кафедра ПМиК

Зав. кафедрой

\_\_\_\_\_\_\_\_\_\_/И.В. Нечта/

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

по направлению 09.03.01 «Информатика и вычислительная техника»

профиль Программное обеспечение средств вычислительной техники и автоматизированных систем

Разработка Android приложения «Угадай картинку”, сгенерированную нейросетью

**Студент**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Шевельков П. С. /

**Институт информатики и вычислительной техники**

**Группа ИП-015**

**Руководитель** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Дементьева К. И./

Новосибирск 2024 г.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc169743710)

[1 обоснование актуальности и ПОСТАНОВКА ЗАДАЧИ 5](#_Toc169743711)

[1.1 ПОСТАНОВКА ЗАДАЧИ 5](#_Toc169743712)

[1.2 АНАЛИЗ СУЩЕСТВУЮЩИХ ПРИЛОЖЕНИЙ 5](#_Toc169743713)

[1.3 ОБОСНОВАНИЕ АКТУАЛЬНОСТИ РАБОТЫ 8](#_Toc169743714)

[1.4 ТРЕБОВАНИЯ К РАЗРАБАТЫВАЕМОМУ ПРИЛОЖЕНИЮ 9](#_Toc169743715)

[2 Выбор средств и инструментов для разработки приложения 10](#_Toc169743716)

[2.1 MODEL 11](#_Toc169743717)

[2.2 VIEW 13](#_Toc169743718)

[2.3 VIEWMODEL 14](#_Toc169743719)

[3 Разработка приложения 16](#_Toc169743720)

[3.1 СОСТАВЛЕНИЕ СХЕМЫ И ДИАГРАММЫ ПРИЛОЖЕНИЯ 16](#_Toc169743721)

[3.2 СОЗДАНИЕ НАВИГАЦИОННОГО ГРАФА ДЛЯ ПРИЛОЖЕНИЯ 16](#_Toc169743722)

[3.3 РЕАЛИЗАЦИЯ MODEL 17](#_Toc169743723)

[3.3.1 Функционал взаимодействия с Firebase 17](#_Toc169743724)

[3.3.2 Написание API-запросов 20](#_Toc169743725)

[3.4 РЕАЛИЗАЦИЯ VIEW 20](#_Toc169743726)

[3.4.1 Разработка фрагментов Авторизация/Регистрация 20](#_Toc169743727)

[3.4.2 Создание фрагмента домашней страницы 23](#_Toc169743728)

[3.4.3 Создание фрагмента профиль 25](#_Toc169743729)

[3.4.4 Создание фрагмента справка 26](#_Toc169743730)

[3.4.5 Создание фрагмента выбора категории 27](#_Toc169743731)

[3.4.5 Создание фрагмента с категориями игры 28](#_Toc169743732)

[3.4 РЕАЛИЗАЦИЯ VIEWMODEL 32](#_Toc169743733)

[3.5.1 Реализация Authorization и Registrtion ViewModel 32](#_Toc169743734)

[3.5.2 Реализация HomeViewModel 32](#_Toc169743735)

[3.5.3 Реализация StartGameViewModel 34](#_Toc169743736)

[3.5.4 Реализация CountriesViewModel 35](#_Toc169743737)

[ЗАКЛЮЧЕНИЕ 38](#_Toc169743738)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 39](#_Toc169743739)

ВВЕДЕНИЕ

В настоящее время нейросети стали неотъемлемой частью жизни многих людей, проникая в самые различные сферы деятельности. Их значительное присутствие обусловлено способностью оптимизировать различные процессы благодаря высокой скорости обработки и анализа информации. В частности, нейросети с успехом применяются для решения задач в области компьютерного зрения, обработки естественного языка, анализа данных и генерации контента.

Одним из интересных и актуальных направлений в развитии нейросетей является область генерации изображений по запросу. Это направление открывает перед нами возможность создания нового контента, соответствующего индивидуальным потребностям и запросам пользователей. Именно в этом контексте оправдывается актуальность проведения исследований и создание проекта, направленного на разработку приложения, в котором будет использоваться нейросеть.

Мобильные приложения играют ключевую роль в нашей повседневной жизни, предоставляя удобный доступ к разнообразным услугам и развлечениям. Они охватывают все аспекты нашей жизни – от общения и социальных сетей до здоровья, финансов и образования. Однако особое место занимают развлекательные приложения. Игры, платформы для просмотра видео и прослушивания музыки, а также приложения для создания и редактирования контента стали неотъемлемой частью современного досуга.

Таким образом, исследование и разработка мобильных приложений, использующих нейросети для генерации контента, представляют собой перспективное направление. Они способны значительно обогатить развлекательную индустрию, предоставляя пользователям возможности для творчества и самовыражения.

Работа направлена на создание уникального приложения на базе Android, в которое будет интегрирована нейросеть, а также реализацию различных функций и пользовательского интерфейса.

# 1 обоснование актуальности и ПОСТАНОВКА ЗАДАЧИ

## ПОСТАНОВКА ЗАДАЧИ

Целью данного проекта является разработка мобильного приложения, которое использует нейросеть для генерации уникальных картинок, созданных по запросам, например, фильмы, пословицы, страны с дальнейшим выводом их на экран. Пользователь должен с нескольких попыток угадать то, что изображено на картинке.

Приложение должно быть интуитивно понятным и легко доступным для широкого круга пользователей.

Основные задачи проекта включают:

* Анализ пользовательских потребностей и предпочтений: определение ключевых функций и возможностей, которые пользователи ожидают от приложения.
* Анализ существующих решений генерации изображений.
* Интеграция нейросети в приложение.
* Создание макета интерфейса приложения.
* Разработка основного функционала приложения и его интерфейса.
* Тестирование и отладка готового проекта.

В результате реализации данного проекта должно быть создано уникальное мобильное приложение, которое использует нейросеть в качестве основного инструмента создания картинок.

## АНАЛИЗ СУЩЕСТВУЮЩИХ ПРИЛОЖЕНИЙ

В последнее время набирают популярность Android-приложения, использующие возможности искусственного интеллекта для распознавания и интерпретации изображений, сгенерированных нейросетью. В данном разделе мы рассмотрим существующие аналоги подобных приложений, оценим их функциональные возможности, пользовательский интерфейс.

1. Wordalle[1]

Наиболее схожее с тематикой моего проекта приложение — Wordalle (Рисунок 1.1), которое объединяет в себе популярную головоломку Wordle[2] и генератор изображений по текстовому запросу DALL-E[3] Mini. В Wordalle действуют основные правила Wordle. Игрокам нужно угадывать, что изображено на сгенерированной картинке, подставляя буквы в поля. Из недостатков можно выделить интерфейс, неприятный глазу и недостаточно качественные изображение. Вероятно, это связанно с тем, что приложение было выпущено в 2022 году и с тех пор не обновлялось, что является также недостатком.

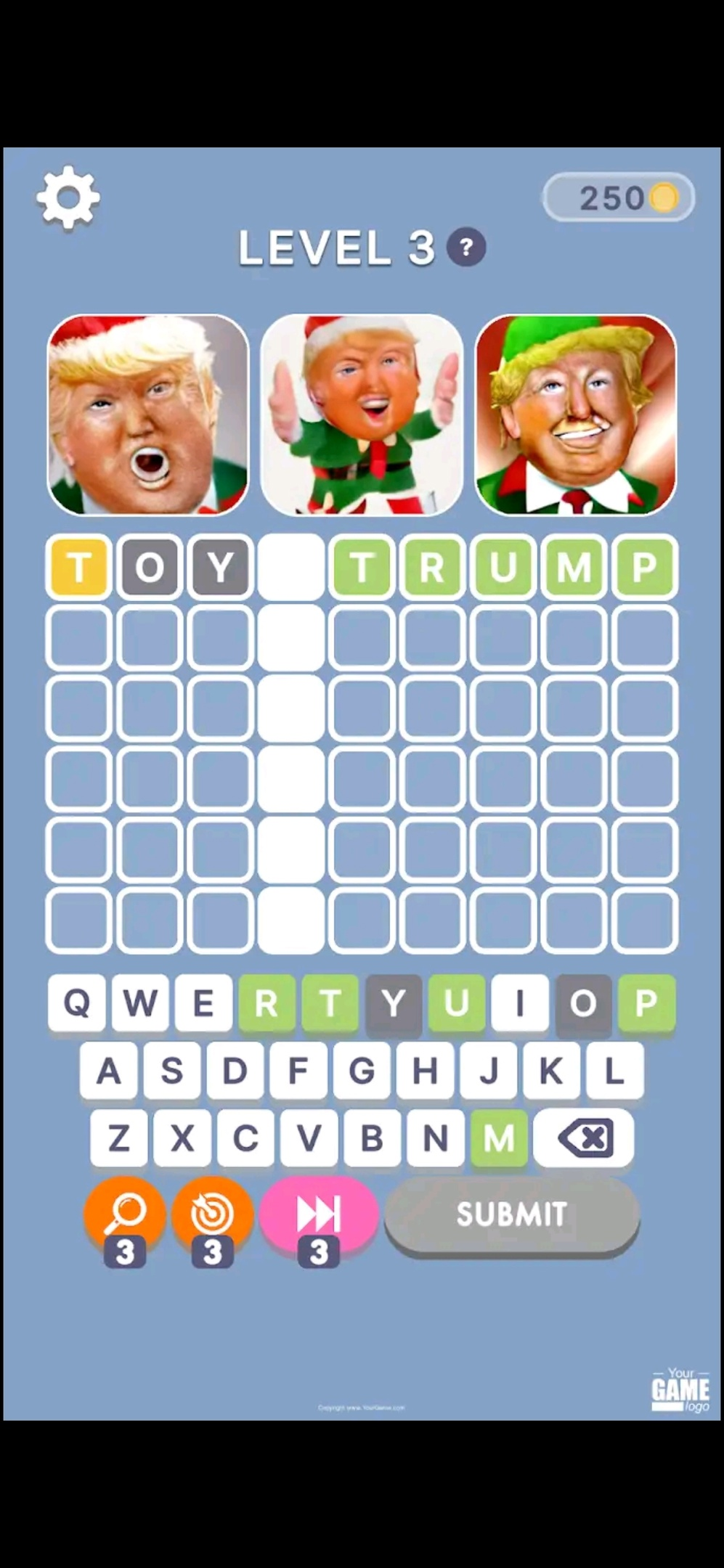


Рисунок 1.1 – Схема навигации приложения

Остальные же платформы больше направлены на обработку сгенерированных либо загруженных с устройства изображений, генерацию по запросу и распознание изображений.

2. DeepArt[4]

DeepArt позволяет пользователям преобразовывать свои фотографии в художественные произведения, используя стили известных художников и методы глубокого обучения. Приложение обрабатывает изображения на серверах, предоставляя высококачественные результаты. Из недостатков: задержки в обработке из-за серверных вычислений, платные функции и ограничения в бесплатной версии.

3. Artbreeder[5]

Artbreeder позволяет создавать новые изображения путем комбинирования существующих. Пользователи могут смешивать различные изображения, изменять параметры и получать уникальные результаты. Недостатками также являются ограничения бесплатной версии и требовательность к ресурсам, что может вызывать задержки на менее мощных устройствах.

4. DALL-E

DALL-E – это продукт компании OpenAI, который использует генеративно-состязательные сети (GAN) для создания изображений на основе текстового описания. Пользователи могут задавать запросы в виде фраз или предложений, после чего DALL-E автоматически создает соответствующие изображения. Этот сервис демонстрирует потенциал в области генерации контента по запросу и предоставляет пользователю инструменты для творческого использования и вдохновения. Недостатками данного продукта также можно считать необходимость подписки для реализации большего потенциала этой платформы и высокие вычислительные ресурсы.

На основе изученных недостатков существующих платформ, таких как высокая требовательность к вычислительным ресурсам, ограничения бесплатного использования и неудобный интерфейс. Приложение должно стремиться к обеспечению локальной обработки данных для повышения скорости и предлагать сбалансированный набор бесплатных функций с возможностью гибкого расширения, а также реализации интуитивного понятного и современного интерфейса.

## ОБОСНОВАНИЕ АКТУАЛЬНОСТИ РАБОТЫ

Разработка мобильных приложений, интегрирующих нейросети, приобретает все большее значение в современной цифровой среде. В частности, использование нейросетей для генерации изображений в развлекательных приложениях открывает новые возможности для взаимодействия и вовлечения пользователей. Актуальность данного проекта обусловлена несколькими важными факторами:

Рост популярности ИИ:

Нейросети демонстрируют выдающиеся результаты в различных областях, включая компьютерное зрение и обработку изображений. Интеграция таких технологий в мобильные приложения позволяет создавать инновационные продукты, которые способны привлекать внимание широкой аудитории и удовлетворять потребности пользователей в уникальном контенте.

Развлекательный аспект:

Современные пользователи стремятся к интерактивным и увлекательным способам времяпрепровождения. Приложение, в котором пользователи могут угадывать, что изображено на картинках, сгенерированных нейросетью, не только обеспечивает развлекательную ценность, но и способствует развитию когнитивных навыков и креативного мышления.

Интерактивность и вовлечение:

Приложения, предлагающие пользователям активное участие, как правило, вызывают больший интерес и удерживают внимание дольше. Возможность угадывать изображения по категориям добавляет элемент игры и соревнования, что повышает уровень вовлеченности и мотивации пользователей.

Актуальность в образовательной сфере:

Развивающие и обучающие элементы приложения могут быть полезны в образовательном контексте. Угадывание изображений может способствовать обучению и тренировке визуального восприятия, а также развивать ассоциативное мышление.

Таким образом, разработка мобильного приложения для платформы Android, которое интегрирует нейросеть для генерации изображений и предоставляет пользователям возможность угадывать их содержание по категориям, является актуальной задачей. Этот проект сочетает в себе современные технические достижения и удовлетворяет потребности пользователей в интерактивных и образовательных развлечениях, что делает его значимым и перспективным в контексте текущих тенденций в области мобильных технологий и искусственного интеллекта.

## ТРЕБОВАНИЯ К РАЗРАБАТЫВАЕМОМУ ПРИЛОЖЕНИЮ

С помощью всего вышесказанного, можно выделить основные требования к приложению:

Дизайн приложения должен быть интуитивно понятен пользователю, чтобы избежать трудностей во время его использования.

Разработка приложения должна быть реализована на платформе Android, чтобы охватить большую часть пользователей

В приложении должен быть реализован основной функционал, такой как регистрация пользователя в системе, для дальнейшего ведения статистки и подсчета прогресса, угадывание картинок, сгенерированных нейросетью.

# 2 Выбор средств и инструментов для разработки приложения

Для разработки android приложения необходимо определиться со стеком технологий, которые будут использоваться для эффективного достижения поставленных целей. В данном разделе представлены ключевые средства разработки.

Общий стек технологий:

* **Android Studio:** Официальная интегрированная среда разработки (IDE) для создания приложений под Android. Обеспечивает широкий набор инструментов для отладки и тестирования приложений.
* **Git:** Инструмент управления версиями для контроля изменений в коде проекта.
* **Gradle:** Система сборки проектов, используемая в Android Studio для управления зависимостями и компиляции приложения.
* **Эмуляторы Android:** Инструменты для тестирования приложения на разных версиях Android и различных устройствах.
* **Firebase[6]:** Набор инструментов и сервисов от Google для разработки мобильных и веб-приложений. Предоставляет возможность быстро развернуть бэкенд с серверной логикой, подключить базы данных и настроить авторизацию пользователей.
* **Kotlin:** Официальный язык программирования для разработки Android-приложений, предоставляющий удобство и безопасность типов.
* **Retrofit[7]:** Библиотека для работы с сетевыми запросами в Android, облегчающая взаимодействие с API и обработку данных.
* **XML:** Язык разметки для описания пользовательского интерфейса в Android, используемый для создания макетов экранов приложений.

Так как в проекте реализована архитектура MVVM стек технологий можно поделить на три раздела: Model, View, ViewModel.

## 2.1 MODEL

Для реализации раздела model, который отвечает за получение и запись данных, необходимых пользователю, в процессе использования приложения, была интегрирована облачная база данных — Firebase от google. Firebase обладает следующими преимуществами:

1. Широкий спектр сервисов:

* Firebase Authentication для упрощенной аутентификации пользователей.
* Cloud Firestore и Realtime Database для хранения данных в реальном времени и синхронизации.
* Firebase Storage для хранения и доставки пользовательских файлов, таких как изображения и видео.
* Firebase Cloud Messaging (FCM) для отправки уведомлений и сообщений.
* Firebase Hosting для быстрой и безопасной развертки веб-приложений.

2. Интеграция с другими сервисами Google:

Firebase легко интегрируется с другими продуктами Google, такими как Google Analytics, Google Ads и Google Cloud Platform. Это позволяет использовать мощные аналитические и рекламные инструменты для улучшения и продвижения приложения.

3. Автоматическое масштабирование

Firebase автоматически масштабируется в зависимости от нагрузки на ваше приложение. Вам не нужно беспокоиться о настройке серверов или управлении инфраструктурой. Это особенно полезно для приложений с переменной нагрузкой.

4.Реальное время

Сервисы Firebase, такие как Realtime Database и Firestore, предоставляют возможность синхронизации данных в реальном времени. Это значит, что любые изменения в данных сразу же отображаются на всех подключенных устройствах, что идеально подходит для чатов, многопользовательских игр и других приложений, требующих мгновенного обновления данных.

5. Простота использования

Firebase предлагает простой и понятный интерфейс, хорошо документированную API и множество примеров и руководств. Это позволяет разработчикам быстро начать работу и сократить время разработки.

6. Безопасность

Firebase предоставляет множество встроенных механизмов безопасности, таких как Firebase Authentication для аутентификации пользователей и правила безопасности Firebase для управления доступом к данным. Это помогает защитить данные пользователей и приложение от несанкционированного доступа.

Таким образом Firebase идеально подходит для написания бэкэнда, включающего в себя авторизацию, ведение статистики и мгновенное обновление данных.

Также для внедрения нейронной модели, способной генерировать изображения по запросу, использовалась библиотека Retrofit, которая упрощает работу с сетевыми запросами в приложениях на Android. С её помощью были разработаны API-запросы для взаимодействия с моделью Kandinsky, доступной в открытом доступе.

## 2.2 VIEW

Для реализации раздела View, который отвечает за отображение данных и взаимодействие с пользователем, была использована библиотека UI-компонентов, описываемая с помощью XML-макетов. Эта библиотека предоставляет следующие преимущества:

* Интуитивный дизайн: Библиотека предоставляет готовые стили и компоненты. Это позволяет создавать приложения с профессиональным и современным внешним видом без необходимости в разработке пользовательского интерфейса с нуля.
* Адаптивность: UI-компоненты автоматически адаптируются под различные размеры экранов и разрешения.
* Поддержка анимаций и переходов: Библиотека предоставляет возможность легко интегрировать анимации и переходы между экранами.
* Гибкость и расширяемость: Возможность кастомизации компонентов позволяет разработчикам адаптировать UI под уникальные требования приложения и бренда, сохраняя при этом высокий уровень производительности и оптимизации интерфейса.

Использование UI-компонентов значительно упрощает процесс разработки пользовательского интерфейса, позволяя сосредоточиться на функциональных возможностях приложения.

## 2.3 VIEWMODEL

Для реализации раздела ViewModel в архитектуре MVVM, который отвечает за управление данными и бизнес-логикой, использовались следующие ключевые технологии и инструменты:

* Kotlin: Официальный язык программирования для Android, который эффективно поддерживает современные практики разработки, такие как асинхронное программирование и функциональное программирование.
* LiveData: Компонент архитектуры Android Jetpack, предоставляющий возможность передачи данных между слоями приложения и автоматического уведомления View о изменениях в данных.
* Coroutines: Библиотека для асинхронного и многопоточного программирования в Kotlin. Она упрощает управление параллельными задачами и обеспечивает безопасное взаимодействие с UI-потоком.
* ViewModel: Часть библиотеки Android Jetpack, предназначенная для хранения и управления данными, необходимыми для UI-компонентов при сохранении состояния при повороте экрана или других изменениях конфигурации.
* Retrofit: Библиотека для работы с сетевыми запросами, используемая в ViewModel для взаимодействия с сервером и получения данных.
* LiveData и Data Binding: Использование LiveData в связке с Data Binding позволяет ViewModel обновлять данные в UI автоматически при изменении данных в ViewModel.

Эти технологии совместно обеспечивают эффективную реализацию ViewModel, разделяющей бизнес-логику и предоставляющей данные для отображения в пользовательском интерфейсе приложения на платформе Android.

# 3 Разработка приложения

## 3.1 СОСТАВЛЕНИЕ СХЕМЫ И ДИАГРАММЫ ПРИЛОЖЕНИЯ

Для успешного старта разработки приложения была подготовлена схема для навигации (Рисунок 3.1).

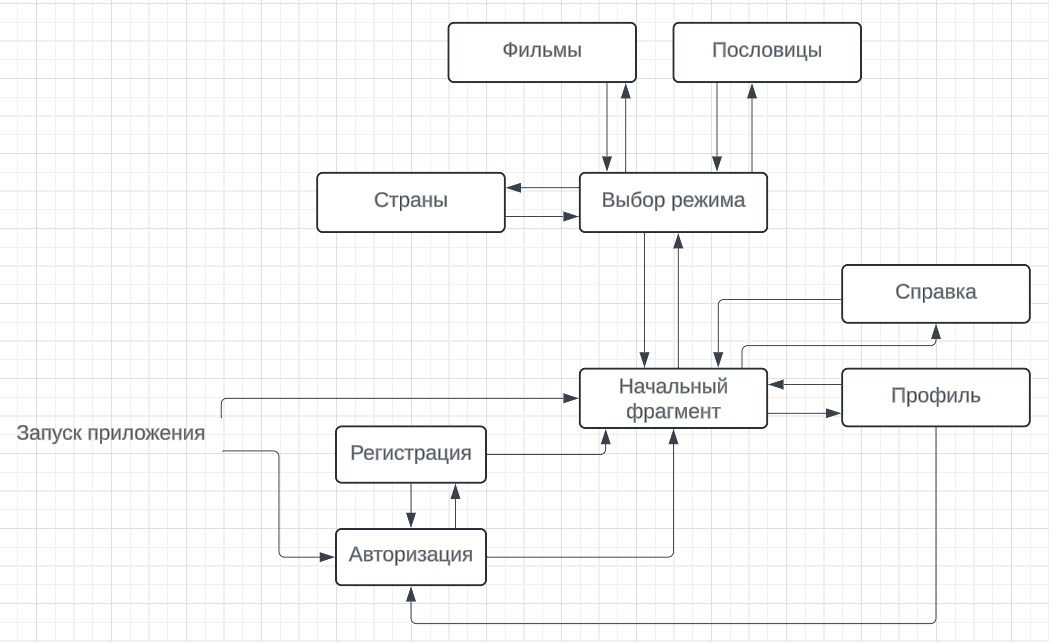


Рисунок 3.1 – Схема навигации приложения

## 3.2 СОЗДАНИЕ НАВИГАЦИОННОГО ГРАФА ДЛЯ ПРИЛОЖЕНИЯ

Для корректной смены фрагментов, необходима реализация навигационного графа, с добавлением всех возможных переходов между фрагментами (Рисунок 3.2).



Рисунок 3.2 – Навигационный граф

Спроектированный навигационный граф соответствует схеме навигации, созданной ранее.

## РЕАЛИЗАЦИЯ MODEL

### 3.3.1 Функционал взаимодействия с Firebase

В ходе интеграции Firebase в проект были разработаны функции: авторизации, регистрации, проверки авторизации, выхода из учётной записи, получения данных из Firebase, обновления данных в Firebase(Рисунок 3.3).

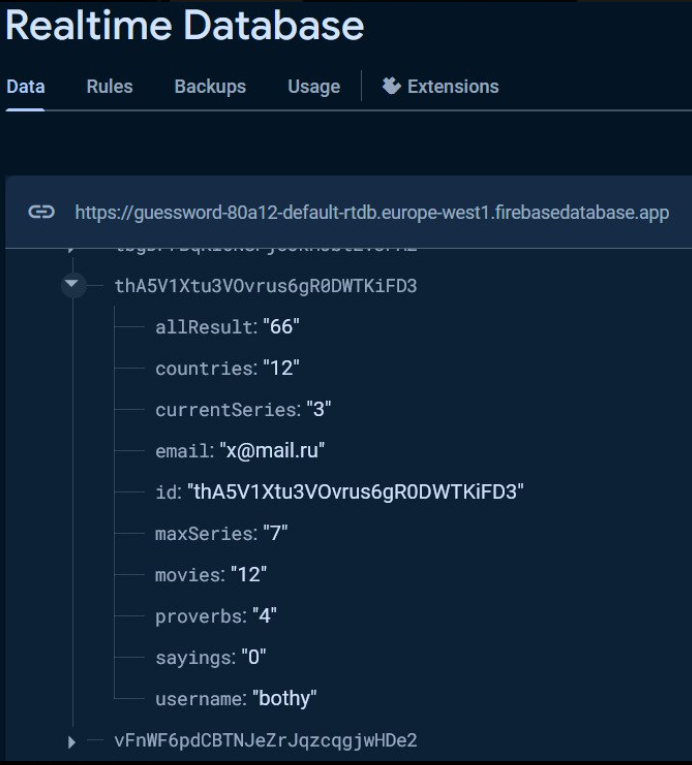


Рисунок 3.3 – База данных со статистикой

Листинг 3.1 — Авторизация

override suspend fun signInWithEmailAndPassword(email: String, password: String, username: String): AuthResult {

return try {

val user = auth.signInWithEmailAndPassword(email, password).await().user!!

AuthResult.Success(User.Base(user.email ?: " ", user.uid, ""))

} catch (e: Exception) {

AuthResult.Error(e)

}

}

Листинг 3.2 — Регистрация  
override suspend fun signUpWithEmailAndPassword(email: String, password: String, username: String ): AuthResult {

return try {

val authResult = auth.createUserWithEmailAndPassword(email, password).await()

val user = authResult.user!!

val userRef = FirebaseDatabase.getInstance().getReference("users").child(user.uid)

val userData = UserSignUp.Base(

email = email, id = user.uid, username = username, allResult = "0", maxSeries = "0", currentSeries = "0", countries = "0", movies = "0", proverbs = "0", sayings = "0"

)

userRef.setValue(userData).await()

AuthResult.SuccesUser(userData)

} catch (e: Exception) {

AuthResult.ErrorUser(e)

}

}

Листинг 3.3 — Выход из учетной записи

override fun logout() {

auth.signOut()

}

Листинг 3.4 — Полученние данных из Firebase

override suspend fun getUserData(): UserData? {

val user = auth.currentUser ?: return null

val userRef = FirebaseDatabase.getInstance().getReference("users").child(user.uid)

return userRef.get().await().getValue(UserData::class.java)

}

Листинг 3.5 — Обновление данных в Firebase

override suspend fun updateUserData(updatedUserData: UserData) {

try {

val user = auth.currentUser ?: throw Exception("User not authenticated")

val userRef = FirebaseDatabase.getInstance().getReference("users").child(user.uid)

val userDataMap = mapOf(

"email" to updatedUserData.email, "id" to updatedUserData.id, "username" to updatedUserData.username, "allResult" to updatedUserData.allResult, "maxSeries" to updatedUserData.maxSeries, "currentSeries" to updatedUserData.currentSeries, "countries" to updatedUserData.countries, "movies" to updatedUserData.movies, "proverbs" to updatedUserData.proverbs, "sayings" to updatedUserData.sayings

)

userRef.updateChildren(userDataMap).await()

} catch (e: Exception) {

throw Exception("Failed to update user data", e)

}

}

### 3.3.2 Написание API-запросов

Для возможности отправки запросов, на сайте «fusionbrain.ai/keys/» были сгенерированы Api ключ и secret, которые передаются при любом обращении к сервису.

Для получения изображения в формате base64 были разработаны три последовательных запроса, которые реализуют функционал генерации изображения с помощью нейронной сети: получение доступной модели, отправка запроса на генерацию изображения, проверка статуса генерации изображения.

Листинг 3.6 — Интерфейс API-запросов

interface ApiService {

@GET("key/api/v1/models")

@Headers("X-Key: Key $apiKey", "X-Secret: Secret $secretKey")

fun getModels(): Call<List<Model>>

@Multipart

@POST("key/api/v1/text2image/run")

@Headers("X-Key: Key $apiKey", "X-Secret: Secret $secretKey")

fun generate(

@Part("model\_id") modelId: RequestBody,

@Part("params") params: RequestBody

): Call<Generate>

@GET("key/api/v1/text2image/status/{request\_id}")

@Headers("X-Key: Key $apiKey", "X-Secret: Secret $secretKey")

fun checkGeneration(@Path("request\_id") requestId: String): Call<CheckGenerate>

}

## 3.4 РЕАЛИЗАЦИЯ VIEW

### 3.4.1 Разработка фрагментов Авторизация/Регистрация

При запуске приложения, в случае если пользователь не авторизирован, он попадает на фрагмент авторизации (Рисунок 3.4).



Рисунок 3.4 – Фрагмент авторизации

В данном фрагменте реализованы переходы на домашнюю страницу, в случае удачной авторизации с использованием почты и пароля, и на регистрацию (Рисунок 3.5).

Листинг 3.7 — Авторизация

binding.signIn.setOnClickListener {

val allValidation = inputList.map { it.isValid() }

if (allValidation.all { it }) {

viewModel.sendCredentials(

email = binding.authMail.text(),

password = binding.authPassword.text(),

username = ""

)

}

}



Рисунок 3.5 – Фрагмент регистрации

В этом окне были реализованы четыре поля для заполнения данных

пользователем и кнопка создания учетной записи. Эти четыре поля –

минимальная информация о пользователе, необходимая для работы

приложения. Нажимая на кнопку "Регистрация" в Firebase разделах: Authentication и Realtime Database создаётся пользователь с уникальным id.

Листинг 3.8 — Регистрация

binding.startSignUp.setOnClickListener {

val allValidation = inputList.map { it.isValid() }

if (allValidation.all { it }) {

viewModel.sendCredentials(

email = binding.signUpEmail.text(),

username = binding.signUpUserName.text(),

password = binding.signUpPasswordLayout.text()

)

}

}

Для этих двух фрагментов была реализована проверка корректности введённых данных (Рисунок 3.6).



Рисунок 3.6 – Проверка корректного ввода данных

Листинг 3.9 — Проверка корректного ввода данных

override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {

}

override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {

onTextChanged.invoke()

}

override fun afterTextChanged(s: Editable?) {

}

### 3.4.2 Создание фрагмента домашней страницы

Данный фрагмент является промежуточным для остальных и реализует правильную навигацию в приложении. В нём реализованы переходы в профиль, справку и раздел с категориями (Рисунок 3.7).



Рисунок 3.7 – Фрагмент домашней страницы

Листинг 3.10 — Создание фрагмента домашней страницы

override fun onCreateView(

inflater: LayoutInflater, container: ViewGroup?,

savedInstanceState: Bundle?

): View? {

\_binding = FragmentHomeBinding.inflate(inflater, container, false)

val root: View = binding.root

return root

}

### 3.4.3 Создание фрагмента профиль

В данном фрагменте (Рисунок 3.8) отображается имя пользователя и его статистика по каждому из разделов, в дополнении к этому есть подсчёт общего результата, лучшей серии и текущей серии отгаданных картинок. Все эти данные изменяются в режиме реального времени, то есть при изменении данных в Firebase, пользователь сразу заметит изменения, как и в случае прохождения. Функционал реализован с помощью слушателя – observe.



Рисунок 3.8 – Фрагмент профиля

Также в профиле реализован выход из учетной записи, нажатием на иконку в левом нижнем углу.

Листинг 3.11 — Создание фрагмента профиль

override fun onCreateView(

inflater: LayoutInflater,

container: ViewGroup?,

savedInstanceState: Bundle?

): View {

\_binding = HomeToFragmentProfileBinding.inflate(inflater, container, false)

val root: View = binding.root

return root

}

### 3.4.4 Создание фрагмента справка

В справке добавлена вся необходимая информация для комфортного использования приложения (Рисунок 3.9).

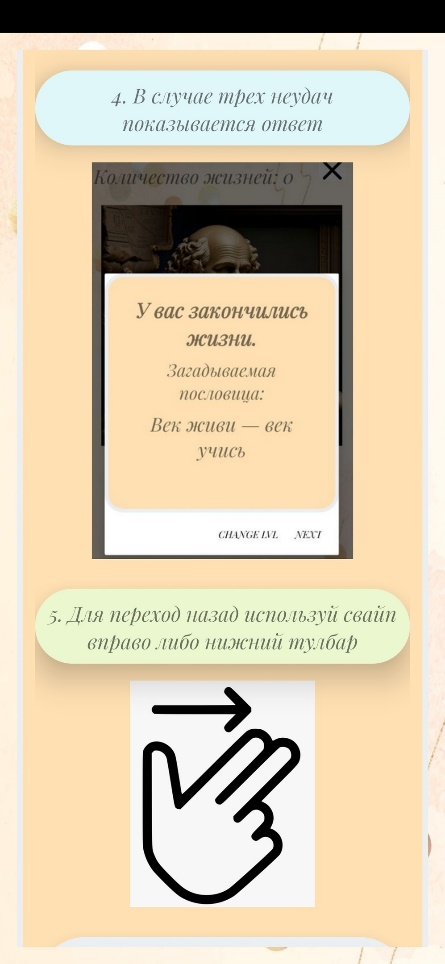
 

Рисунок 3.9 – Фрагмент справка

Листинг 3.12 — Создание фрагмента справка

override fun onCreateView(

inflater: LayoutInflater, container: ViewGroup?,

savedInstanceState: Bundle?

): View? {

val view = inflater.inflate(R.layout.home\_to\_fragment\_reference, container, false)

return view

}

### 3.4.5 Создание фрагмента выбора категории

В данном разделе (Рисунок 3.10) созданы три кнопки с отображением прогресса по соответствующей категории. Прогресс меняется в режиме реально времени, по мере прохождения. Реализовано с помощью слушателя – observe.

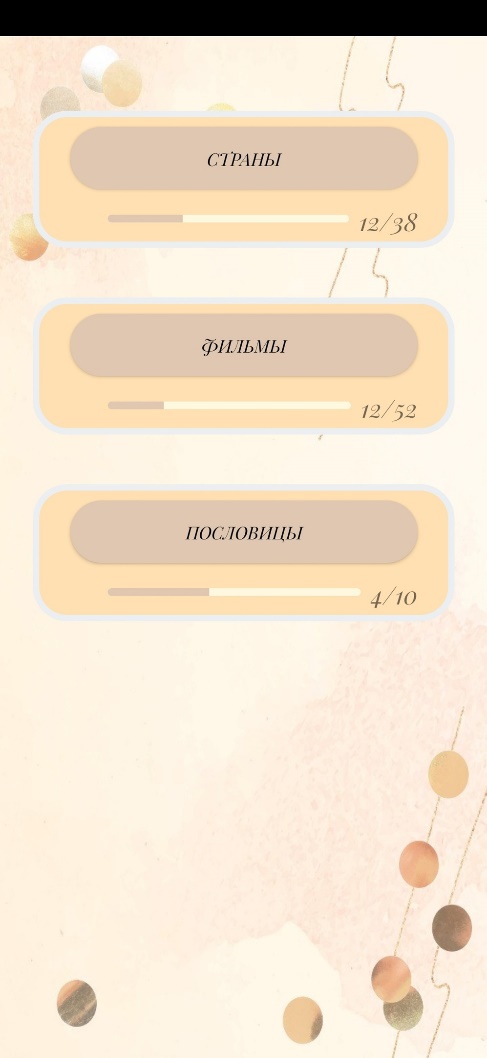


Рисунок 3.10 – Фрагмент выбора категории

Листинг 3.13 — Создание фрагмента выбора категории

override fun onCreateView(

inflater: LayoutInflater,

container: ViewGroup?,

savedInstanceState: Bundle?

): View {

\_binding = HomeToFragmentStartGameBinding.inflate(inflater, container, false)

val root: View = binding.root

return root

}

### 3.4.5 Создание фрагмента с категориями игры

В этих фрагментах (Рисунок 3.11) реализован основной функционал приложения – отгадывание картинок, сгенерированных нейросетью. Изначально отображается прогресс бар – загрузка, означающая процесс генерации изображения. После успешной генерации пользователю необходимо отгадать по изображению, что было сгенерировано нейронной моделью.

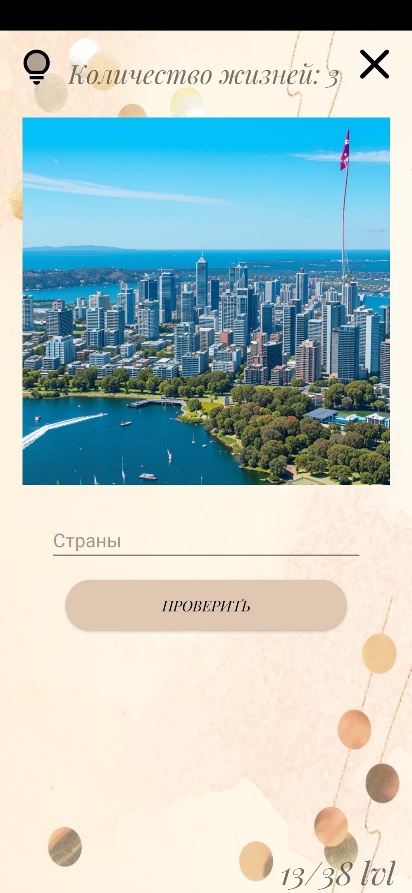
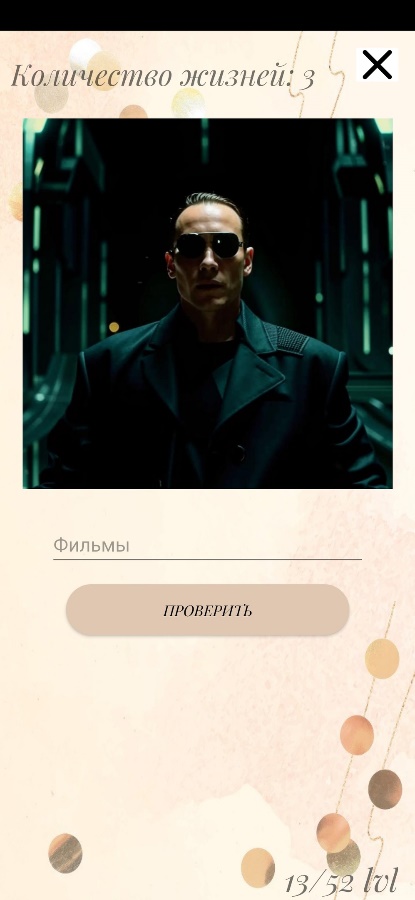
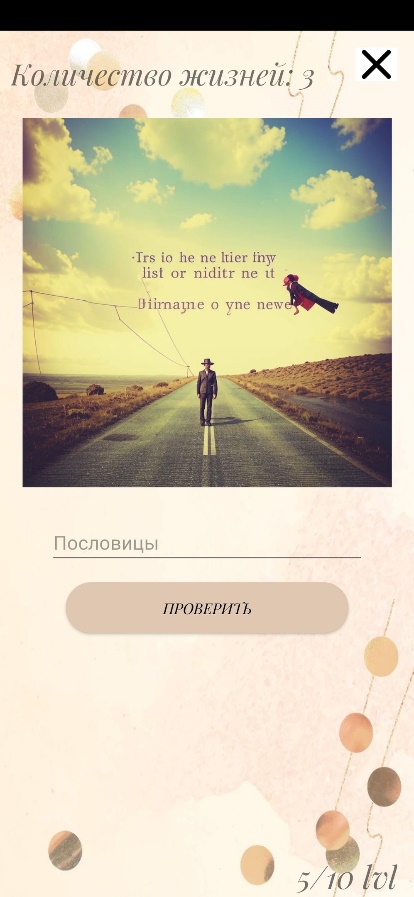
  

Рисунок 3.11 – Фрагменты категорий

Листинг 3.14 — Создание фрагментов категорий

override fun onCreateView(

inflater: LayoutInflater,

container: ViewGroup?,

savedInstanceState: Bundle?

): View {

\_binding = LvlFragmentCountriesBinding.inflate(inflater, container, false)

val root: View = binding.root

return root

}

override fun onCreateView(

inflater: LayoutInflater,

container: ViewGroup?,

savedInstanceState: Bundle?

): View {

\_binding = LvlFragmentMoviesBinding.inflate(inflater, container, false)

val root: View = binding.root

return root

}

override fun onCreateView(

inflater: LayoutInflater,

container: ViewGroup?,

savedInstanceState: Bundle?

): View {

\_binding = LvlFragmentProverbsBinding.inflate(inflater, container, false)

val root: View = binding.root

return root

}

Для данных фрагментов было реализовано диалоговое окно, с отображение результата уровня и кнопками для перехода на следующий уровень, в фрагмент выбора категории (Рисунок 3.12).

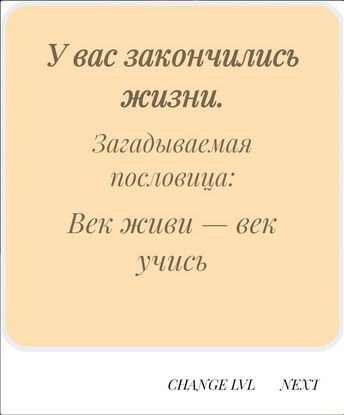
 

Рисунок 3.12 – Диалоговое окно с результатом

Листинг 3.15 – Создание диалогового окна

val builder = AlertDialog.Builder(requireContext())

.setView(dialogView)

.setCancelable(false)

.setPositiveButton("next") { dialog, \_ ->

findNavController().navigate(R.id.action\_proverbsFragment\_self)

dialog.dismiss()

}

.setNegativeButton("change lvl") { dialog, \_ ->

val navController = findNavController()

val navOptions = NavOptions.Builder()

.setPopUpTo(R.id.startGameFragment, true)

.build()

navController.navigate(

R.id.action\_proverbsFragment\_to\_startGameFragment,

null,

navOptions

)

dialog.dismiss()

}

val dialog = builder.create()

dialog.show()

## 3.4 РЕАЛИЗАЦИЯ VIEWMODEL

### 3.5.1 Реализация Authorization и Registrtion ViewModel

Для данных ViewModel были разработаны функции, которые получают результат от функций регистрации и авторизации, созданных в разделе Model.

ViewModel получает данные от Model, а View слушает изменения ViewModel и отображает их пользователю.

Листинг 3.16 получение результата авторизации и регистрации  
@HiltViewModel  
class AuthorizationViewModel @Inject constructor(private val authRepository: AuthRepository) :  
 BaseViewModel() {  
 override val sendRequest: suspend (String, String, String) -> AuthResult =  
 **{** email, password, username **->** authRepository.signInWithEmailAndPassword(email, password, username) **}**}

@HiltViewModel  
class RegistrationViewModel @Inject constructor(private val authRepository: AuthRepository) :  
 BaseViewModel() {  
  
 override val sendRequest: suspend (String, String, String) -> AuthResult =  
 **{** email, password, username **->** authRepository.signUpWithEmailAndPassword(email, password, username) **}**}

### 3.5.2 Реализация HomeViewModel

В данном классе разработаны функции для получения данных, изменения статистики по мере прохождения. При создании его экземпляра, путем инициализации была реализована загрузка всех данных пользователя в переменные HomeViewModel.

Листинг 3.17 Реализация получения данных пользователя

private fun loadData() {  
 *CoroutineScope*(Dispatchers.IO).*launch* {  
 val userData = authRepository.getUserData()  
 \_userName.postValue(userData?.username)  
 \_allResult.postValue(userData?.allResult)  
 \_currentSeries.postValue(userData?.currentSeries)  
 \_maxSeries.postValue(userData?.maxSeries)  
 val currentCountries = (userData?.countries?.*toIntOrNull*() ?: 0)  
 val allCountries = DataStore.countries.size  
 \_countUserCountries.postValue(currentCountries)  
 \_allCountries.postValue(allCountries)  
 \_progressCountries.postValue(calculatePercentage(currentCountries, allCountries))  
 val currentMovies = (userData?.movies?.*toIntOrNull*() ?: 0)  
 val allMovies = DataStore.movies.size  
 \_countUserMovies.postValue(currentMovies)  
 \_allMovies.postValue(allMovies)  
 \_progressMovies.postValue(calculatePercentage(currentMovies, allMovies))  
 val currentProverbs = (userData?.proverbs?.*toIntOrNull*() ?: 0)  
 val allProverbs = DataStore.proverbs.size  
 \_countUserProverbs.postValue(currentProverbs)  
 \_allProverbs.postValue(allProverbs)  
 \_progressProverbs.postValue(calculatePercentage(currentProverbs, allProverbs))  
 }  
}

Листинг 3.18 изменения статистики при поражении

fun updateUserDataForLose(nameGame: String) {  
 *CoroutineScope*(Dispatchers.IO).*launch* {  
 val userData = authRepository.getUserData()  
 userData?.*let* {  
 val updatedUserData = it.copy(  
 countries = if (nameGame == "countries") ((it.countries?.*toIntOrNull*() ?: 0) + 1).toString() else it.countries,  
 movies = if (nameGame == "movies") ((it.movies?.*toIntOrNull*() ?: 0) + 1).toString() else it.movies,  
 proverbs = if (nameGame == "proverbs") ((it.proverbs?.*toIntOrNull*() ?: 0) + 1).toString() else it.proverbs,  
 allResult = ((it.allResult?.*toIntOrNull*() ?: 0) + 1).toString(),  
 currentSeries = "0"  
 )  
 authRepository.updateUserData(updatedUserData)  
 }  
 }  
}

Листинг 3.19 изменения статистики при угадывании

fun updateUserDataForWin(nameGame: String) {  
  
 *CoroutineScope*(Dispatchers.IO).*launch* {  
 val userData = authRepository.getUserData()  
 userData?.*let* {  
 val updatedCurrentSeries = (it.currentSeries?.*toIntOrNull*() ?: 0) + 1  
 val updatedMaxSeries = *maxOf*(updatedCurrentSeries, (it.maxSeries?.*toIntOrNull*() ?: 0))  
 val updatedUserData = it.copy(  
 countries = if (nameGame == "countries") ((it.countries?.*toIntOrNull*() ?: 0) + 1).toString() else it.countries,  
 movies = if (nameGame == "movies") ((it.movies?.*toIntOrNull*() ?: 0) + 1).toString() else it.movies,  
 proverbs = if (nameGame == "proverbs") ((it.proverbs?.*toIntOrNull*() ?: 0) + 1).toString() else it.proverbs,  
 allResult = ((it.allResult?.*toIntOrNull*() ?: 0) + 1).toString(),  
 currentSeries = updatedCurrentSeries.toString(),  
 maxSeries = updatedMaxSeries.toString()  
 )  
 authRepository.updateUserData(updatedUserData)  
 }  
 }  
}

### 3.5.3 Реализация StartGameViewModel

В этом классе были созданы следующие функции: обработчик количества жизней, проверка категории на прохождение всех уровней.

Листинг 3.20 обработчик количества жизней

fun decrementLifes() {  
 val currentLifes = \_lifes.*value* ?: 3  
 if (currentLifes > 0) {  
 \_lifes.*value* = currentLifes - 1

Листинг 3.21 проверка категории на прохождение всех уровней

fun checkStartLvlInStartFragment(number: Int, total: Int): Boolean {  
 if (number==total)  
 return false  
 else  
 return true  
}

### 3.5.4 Реализация CountriesViewModel

В данном ViewModel была разработана функция, выполняющая три последовательных запроса, реализованных в Model. В ней были созданы исключения для каждого запроса, а также декодировка полученного изображения из формата base64 в формат BitMap для установления в ImageView.

Листинг 3.22 Получение изображения через API-запросы

fun getImageFromAPI(numOfLvl: Int, nameOfLvl: String, callback: ImageLoadCallback) {  
 val retrofit = Retrofit.Builder()  
 .baseUrl("https://api-key.fusionbrain.ai/")  
 .addConverterFactory(GsonConverterFactory.create())  
 .build()  
 val apiService = retrofit.create(ApiService::class.*java*)  
 val apiHelper = ApiHelper(apiService)  
 *viewModelScope*.*launch* {  
 if (!*isActive*) return@launch   
  
 try {  
 apiHelper.getModels { models ->  
 if (models != null) {  
 Log.d("CountryFragment", "profileViewModel: $numOfLvl")  
 val firstModelId = models.*firstOrNull*()?.id.*toString*()  
  
 val dataName = when (nameOfLvl) {  
 "countries" -> "город в " + DataStore.countries[numOfLvl]  
 "movies" -> DataStore.movies[numOfLvl]  
 else -> DataStore.proverbs[numOfLvl]  
 }  
 apiHelper.sendGenerateRequest(firstModelId, dataName) { generateResponse ->  
 if (generateResponse != null) {  
 val uuidPar = generateResponse.uuid.toString()  
 apiHelper.checkGeneration(apiService, uuidPar, *viewModelScope*) { checkGenerateResponse ->  
 if (checkGenerateResponse != null) {  
 Log.d("LvlFragment", "remainingAttempts: ${checkGenerateResponse[0]}")  
  
 try {  
 @OptIn(kotlin.io.encoding.ExperimentalEncodingApi::class)  
 val decodedBytes: ByteArray = Base64.decode(checkGenerateResponse[0])  
 if (decodedBytes.*isNotEmpty*()) {  
 val decodedBitmap: Bitmap? = BitmapFactory.decodeByteArray(decodedBytes, 0, decodedBytes.size)  
 if (decodedBitmap != null) {  
 \_img.postValue(decodedBitmap)  
 } else {  
 Log.e("CountryFragment", "Failed to decode bitmap")  
 }  
 } else {  
 Log.e("CountryFragment", "Decoded bytes are null or empty")  
 }  
 } catch (e: Exception) {  
 Log.e("CountryFragment", "Error decoding image: ${e.message}")  
 }  
 } else {  
 Log.e("CountryFragment", "Error occurred during checkGenerate request")  
 }  
 }  
 } else {  
 Log.e("CountryFragment", "Error occurred during generate request")  
 }  
 }  
 } else {  
 Log.e("CountryFragment", "No models found")  
 }  
 }  
 } catch (e: Exception) {  
 withContext(Dispatchers.Main) {  
 Log.e("CountryFragment", "Error occurred: ${e.message}")  
 }  
 }  
 }  
}

Для реализации MoviesViewModel и ProverbsViewModel используется такая же функция, как и для CountriesViewModel.

ЗАКЛЮЧЕНИЕ

В ходе работы над данным проектом был использован большой стек технологий, которые помогли создать уникальное приложение с интегрированной нейросетью.

Анализ других приложений с похожей тематикой дал понять то, что должно быть выполнено в работе, чтобы заинтересовать пользователя и сделать процесс игры более увлекательным.

Этот проект объединяет достижения в изучении нейросетей и интерактивные элементы развлекательных приложений, создавая уникальный продукт, который способен привлечь внимание широкой аудитории. Возможность угадывания изображений по различным категориям добавляет способствует развитию когнитивных и креативных навыков пользователей.

Разработка этого приложения требует тщательной проработки алгоритмов генерации изображений, создания интуитивно понятного интерфейса и обеспечения безопасности данных, что в свою очередь способствует развитию профессиональных навыков в области искусственного интеллекта и мобильной разработки.

Таким образом, данный проект представляет собой значимый вклад в развлекательную индустрию и образовательные технологии, открывая новые горизонты для применения нейросетей. Успешная реализация этого приложения демонстрирует перспективы и потенциал современных мобильных решений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Головоломка Wordalle [Электронный ресурс]. URL: https://play.google.com/store/apps/details?id=game.wordalle (Дата обращения 17.05.2024)
2. Головоломка Wordle [Электронный ресурс]. URL: https://play.google.com/store/apps/details?id=one.belousov.wordgame (Дата обращения 17.05.2024)
3. Статья как работает DALL-E [Электронный ресурс]. URL: https://habr.com/ru/companies/ruvds/articles/687508/ (Дата обращения 10.05.2024)
4. Deepart.io — сервис для создания арта на основе фотографий и художественных изображений [Электронный ресурс]. URL: https://www.white-windows.ru/deepart-io-servis-dlya-sozdaniya-arta-na-osnove-fotografij-i-hudozhestvennyh-izobrazhenij/ (Дата обращения 12.05.2024)
5. Artbreeder: как нейросеть создает картинки по скетчам и фэнтези-портреты [Электронный ресурс]. URL: https://trends.rbc.ru/trends/industry/6464d0569a7947a6c7d5a852 (Дата обращения 14.05.2024)
6. Что такое Firebase [Электронный ресурс]. URL: https://kolmogorov.pro/what-is-firebase-chto-takoe (Дата обращения 24.05.2024)
7. Как библиотека Retrofit 2 решает сетевые трудности перевода [Электронный ресурс]. URL: https://practicum.yandex.ru/blog/retrofit-na-android/ (Дата обращения 26.05.2024)