

**МИНОБРНАУКИ РОССИИ**

**Федеральное государственное бюджетное образовательное учреждение  
высшего образования**

**«Чувашский государственный университет имени И.Н.Ульянова»  
(ФГБОУ ВО «ЧГУ им. И.Н.Ульянова»)**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ, ФИЗИКИ И  
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

**КАФЕДРА ДИСКРЕТНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

на тему:

**«РАЗРАБОТКА МОБИЛЬНОГО ПЕРЕВОДЧИКА ТЕКСТОВ ДЛЯ  
ANDROID»**

Студент гр. ФМ-12-21 \_\_\_\_\_ Студнева Карина Артемовна \_\_\_\_\_  
(подпись, Ф.И.О.)

Научный  
руководитель \_\_\_\_\_ к.п.н, доцент Речнов Алексей Владимирович \_\_\_\_\_  
(подпись, учен. степень, учен. звание, Ф.И.О.)

Заведующий  
кафедрой \_\_\_\_\_ к.ф.-м.н., доцент Троешестова Дарья Анатольевна \_\_\_\_\_  
(подпись, учен. степень, учен. звание, Ф.И.О.)

Чебоксары 2025

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	3
ГЛАВА 1. ОСНОВЫ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ .....	6
1.1 Общая концепция мобильных приложений. Классификация мобильных приложений и сопоставление программных инструментов. ....	6
1.2 Подготовка Android Studio к разработке .....	9
1.3 API-сервисы для перевода и анализ баз данных. ....	13
ГЛАВА 2. РАЗРАБОТКА ЭКРАННОГО ПЕРЕВОДЧИКА .....	18
2.1 Создание дизайна переводчика .....	18
2.2 Создание проекта, настройка зависимостей и перенос дизайна. ....	21
2.3 Доработка макета страницы переводчика и реализация функций. ....	25
ЗАКЛЮЧЕНИЕ .....	35
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	37
ПРИЛОЖЕНИЯ .....	40

## ВВЕДЕНИЕ

В двадцать первом веке с появлением глобальной сети Интернет у нас есть возможность поддерживать связь с людьми с разных концов света, изучать их историю, культуру и традиции. Однако препятствует этому языковой барьер. Конечно, можно выучить иностранный язык, чтобы избежать подобных проблем, но на это уйдёт уйма времени. Универсальным и быстрым решением проблемы является переводчик.

Сейчас большинство людей имеют портативные медиа-устройства: смартфон, планшет, ноутбук, компьютер. Они для нас стали привычными и необходимыми для работы, учёбы, досуга и поддержания связи.

Для более оперативного перевода удобно использовать мобильные устройства, такие как смартфоны. В прошлом году лидерами по продажам смартфонов в России остаются бренды выпускающие устройства на ОС Android. Именно эта ОС так популярна, так как примерно 70% всех мобильных устройств в России работают на Android (данные StatCounter на апрель 2025 года). Оперативная система удерживает свое лидерство на рынке мобильных устройств с 2012 года (вот уже 13 лет). Почему это происходит, ведь по словам InMobi, Android появились даже позже iOS. Во-первых, в отличие от других ОС, Android имеет открытый исходный код. Разработанная и выпущенная компанией Google, она открыта для других компаний. Во-вторых, цена также имеет значение. Устройства на Android дешевле, чем на iOS. И хотя цены, как и в любой сфере, растут, они все равно доступнее. На рисунке 1 изображена диаграмма доли рынка мобильных операционных систем во Российской Федерации с апреля 2024 по апрель 2025 года.

В данной ВКР будет описана разработка мобильного переводчика текстов на базе ОС Android. Конечным результатом моей работы будет экранный переводчик с множеством настроек перевода и окна с переведённым текстом.

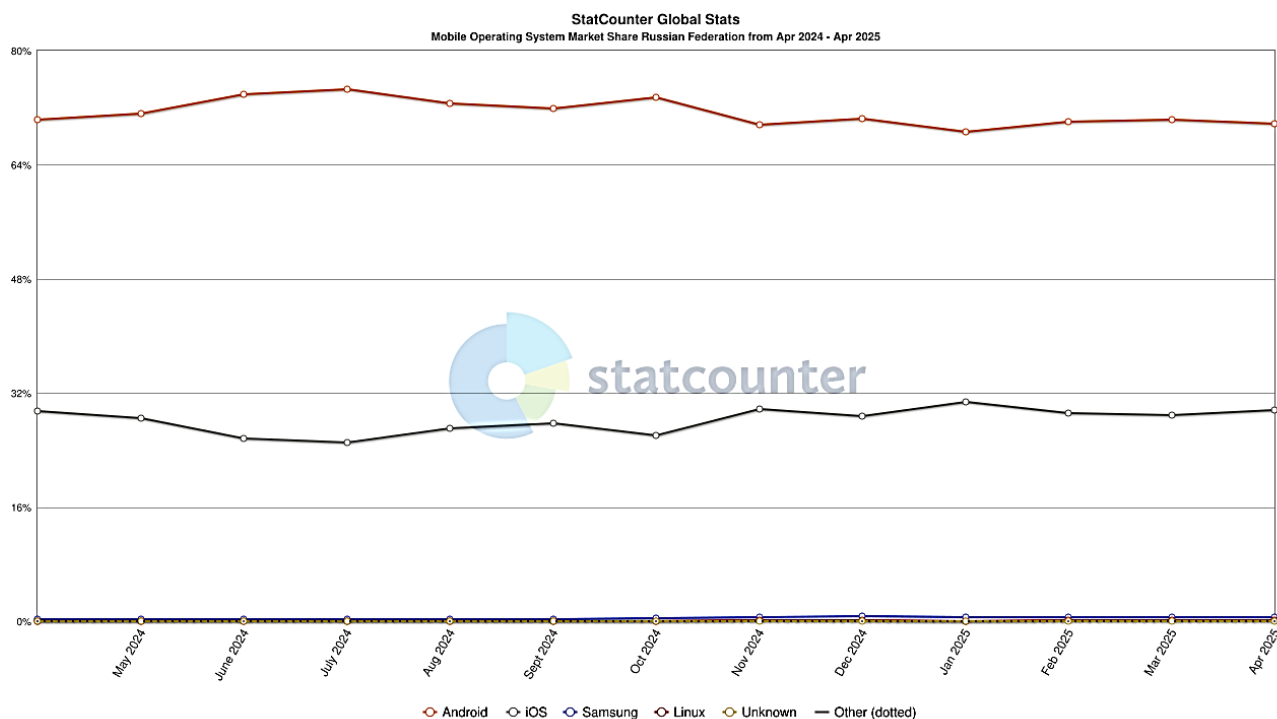


Рисунок 1. Диаграмма доли рынка мобильных операционных систем

Проблемой исследования является отсутствие на рынке бесплатных экранных переводчиков для Android-устройств. Также они либо не имеют нужных функций, либо ограничены в предоставлении перевода. Соответственно, моё приложение решит эти проблемы, что сделает доступным и быстрым перевод текстов с различных источников (книги, статьи, фильмы и т. д.) для многих пользователей. Этим и обусловлена актуальность моей работы.

Процесс разработки экранного переводчика с обширным набором настроек является объектом исследования данной работы. А создание приложения-переводчика для устройств на базе Android ОС с большим количеством настраиваемых функций перевода текстов — предметом исследования.

Цель моей работы — разработка мобильного приложения экранного переводчика текстов для Android-смартфонов, в котором имеются разные настройки самого перевода.

Исходя из цели, мы можем поставить следующие задачи:

1. Провести анализ процесса создания мобильного приложения для ОС Android.
2. Выбор среды разработки для Android.
3. Разработать функционал приложения и его дизайн.
4. Реализовать проект.

# **ГЛАВА 1. ОСНОВЫ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ**

## **1.1 Общая концепция мобильных приложений. Классификация мобильных приложений и сопоставление программных инструментов.**

Мобильное приложение – является программным обеспечением, которое специально разработано для конкретной мобильной платформы. Предназначено для применения на разных портативных устройствах (планшете, умных часах и иных мобильных устройствах).

Чтобы создать мобильное приложение, можно использовать разные языки программирования. К примеру, для Android используют Java и Kotlin, для IOS – Swift и Objective-C.

Приложение на мобильное устройство можно установить через встроенный магазин (Google Play, App Store, и другие).

Процесс создания мобильных приложений протекает следующим образом:

- 1) Написание на языке программирования высокого уровня;
- 2) Компиляция в машинный код операционной системы.

Классификация приложений осложняется тем, что большинство из них являются «гибридными», то есть они предлагают пользовательские функции различных радаров. Однако существуют несколько типов мобильных приложений.

Среди пользователей мобильных приложений набрали свою популярность контент-приложения. На данный момент такие действия, как прослушивание музыки, просмотр фильмов, видеороликов, фотографий и чтение цифровых книг, настолько же доступны и удобны для каждого владельца мобильного устройства, так как это создает спрос на этот сегмент мобильных приложений.

Бизнес-приложение считается неотъемлемым инструментом для многих пользователей, который поможет им ускорить и оптимизировать работу в офисе. В настоящее время для инвесторов и бизнес-компаний, сегмент бизнес-приложений является наиболее удобным, но сложность заключается в переносе бизнес-задач на мобильные устройства.

Мобильные игры сегодня занимают значительную часть на рынке мобильных приложений. Разработчики создают новые игры или улучшают и обновляют уже выпущенные.

Мобильные социальные сети набирают все большую популярность с каждым днем, увеличивая аудиторию по всему миру. Сегодня огромное число людей пользуются социальными сетями, на что влияет еще одна тенденция, которая уже сформировалась: увеличение числа пользователей смартфонов. Можно сказать, что это самый популярный тип мобильных приложений, из-за того, что при покупке устройства там уже установлена одна или несколько мобильных программ по умолчанию.

Разработка экранного приложения-переводчика довольно сложна. Для этого нужно учесть множество факторов: внедрение функций распознавания текста, обработка видео в реальном времени и сокращение задержек вывода.

При разработке приложения используются разные среды разработки - Android Studio, Flutter и Xamarin. Далее рассмотрим преимущества и особенности каждой из вышеперечисленных сред.

Flutter от Google обеспечивает высокую скорость создания интерфейсов благодаря графическому движку Skia. Это позволяет отображать текст на видео с камеры с частотой 60 кадров в секунду, что особенно важно для дополненной реальности. На платформе pub.dev доступны оптимизированные библиотеки: `google_ml_kit` для распознавания текста на 98 языках с точностью до 93% и `camera` для управления настройками экспозиции и фокусировки.

Однако при работе с более сложными задачами, например, с переводом рукописного текста с изогнутых поверхностей, Flutter сталкивается с ограничениями Dart. Операции с изображениями выполняются на 40%

медленнее, чем нативный код. Это требует использования платформенно-специфичных модулей на Java/Kotlin через platform channels, что усложняет отладку и увеличивает время разработки.

Xamarin от Microsoft представляет собой гибридную модель, сочетающую бизнес-логику на C# с нативными вызовами API Android. Это позволяет использовать Microsoft Azure Cognitive Services, такие как Form Recognizer для извлечения текста из документов и Custom Translator для обучения моделей на специализированной лексике. Для защиты данных применяются встроенные механизмы .NET, включая шифрование AES-256 через System.Security.Cryptography.

Однако архитектурные компромиссы приводят к задержкам. Обработка HD-кадра через JNI занимает до 120 миллисекунд, в то время как нативный код в Android Studio — всего 60-70 миллисекунд. Размер APK также увеличивается из-за включения библиотек .NET и среды выполнения Mono, достигая 90-110 мегабайт.

Android Studio остаётся оптимальным выбором для задач, требующих аппаратной оптимизации. Android NDK позволяет создавать производительные модули на C++. Например, для распознавания текста можно использовать OpenCV для бинаризации и сегментации символов, Tesseract-OCR с предобученной LSTM-сетью и ICU для нормализации юникода. Camera2 API в сочетании с GPU-ускоренной обработкой и выводом через Vulkan обеспечивает задержки  $\leq 50$  миллисекунд на флагманских устройствах. ML Kit позволяет кастомизировать TensorFlow Lite-модели, обучая их на специфических шрифтах или уменьшая размер модели. Оптимизированный пайплайн в Android Studio обрабатывает страницу формата A4 со смешанными языками за 0,7 секунды, в то время как Flutter — за 1,2 секунды.

Энергетическая эффективность также важна. Тестирование на Samsung Galaxy S23 показало следующие результаты: Android Studio потребляет 8-10 мА/ч при использовании камеры и машинного обучения; Xamarin — 12-15



мА/ч из-за JNI-вызовов; Flutter — 14-18 мА/ч при рендеринге сложных анимаций.

Интеграция с периферийными устройствами также различается. Только Android Studio поддерживает Hardware Buffer API, что позволяет передавать данные с камеры в память графического процессора без копирования. Xamarin требует обёрток для работы с устройствами, такими как C-Pen, а Flutter ограничен в использовании TOF-камер для определения расстояния до текста.

Для создания эффективного переводчика с технологией дополненной реальности (AR) в рамках дипломной работы использование Android Studio предоставляет ряд значимых преимуществ:

1. Высокая скорость работы системы благодаря прямому доступу к Camera2 API, NDK и специализированным ускорителям машинного обучения (ML).
2. Энергоэффективность, что особенно важно для мобильных приложений.
3. Гибкая настройка моделей машинного обучения под конкретные сценарии использования.

## **1.2 Подготовка Android Studio к разработке**

Подготовить Android Studio к разработке не трудно. Нужно для начала скачать и установить Android Studio, Android SDK и эмулятор Android.

Для начала следует установить программу Android Studio с официального сайта, подобрав его к своей операционной системе. Установку можно произвести как через установочный файл .exe, так и через архив .zip, отличие лишь в необходимости ручной настройки многих компонентов при загрузке .zip файла.

Далее следует скачать Java Development Kit (JDK). Для этого перейдём на сайт Oracle и установим нужную нам версию JDK. Также обязательно укажем

путь к установленной JDK в самом Android Studio (File → Project Structure → SDK Location).

При первом запуске Android Studio: выбираем путь установки самого приложения, есть возможность настроить внешний вид IDE (Integrated Development Environment – интегрированной среды разработки). Далее идёт выбор компонентов установки, выбираем нужные и ждём окончательной загрузки всех компонентов Android Studio.

Наконец, при правильной установке, открываем нашу IDE и создаем новый проект – «Start a new Android Studio project». В мастере создания выбираем шаблон для нашего будущего приложения. Рекомендуется использовать «Empty Activity» или «Empty Views Activity».

В чём же их различие? «Empty Activity» по умолчанию использует в виде языка для разработки Kotlin, а «Empty Views Activity» позволяет выбрать язык программирования – Java или Kotlin. В данной работе мной был выбран «Empty Views Activity».

После перед нами открывается окно настроек проекта (Рисунок 2).

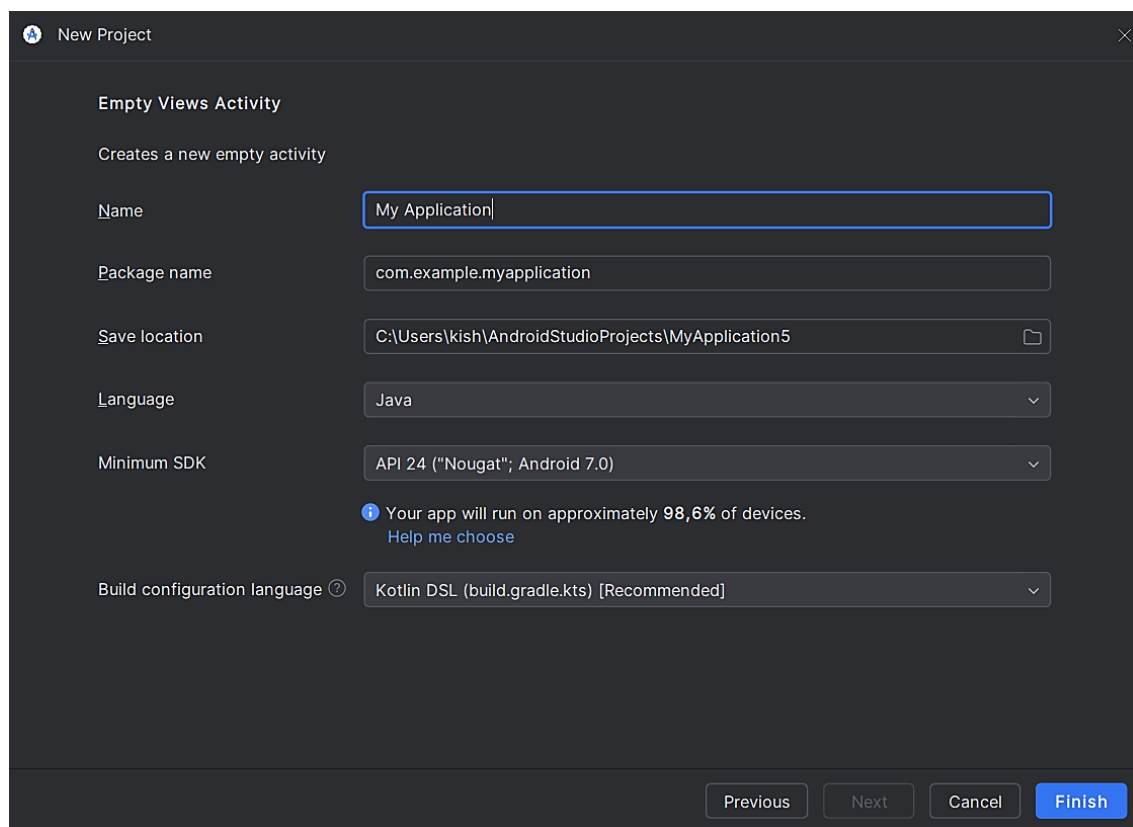


Рисунок 2. Окно настроек проекта Android Studio.

Поле «Name» служит для названия проекта, имени приложения.

Поле «Package name» указывает на уникальное имя пакета, где будет размещен главный класс приложения.

В поле «Save Location» разработчик устанавливает расположение файлов проекта.

Поле «Language» позволяет выбрать язык программирования. В данном проекте выбран язык Java.

В поле «Minimum SDK» разработчик указывает версию SDK. Чем меньше будет версия SDK, тем больше устройств будет поддерживать данное мобильное приложение.

В поле «Build configuration language» указывается язык, используемый для конфигурации системы сборки проекта. Сейчас Kotlin лучше интегрируется с редактором кода Android Studio, чем Groovy. Но сборки с использованием Kotlin обычно медленнее выполняются, чем сборки с Groovy.

После нажатия кнопки «Finish», Android Studio создает новый проект. Далее вы можете рассмотреть структуру проекта (Рисунок 3).

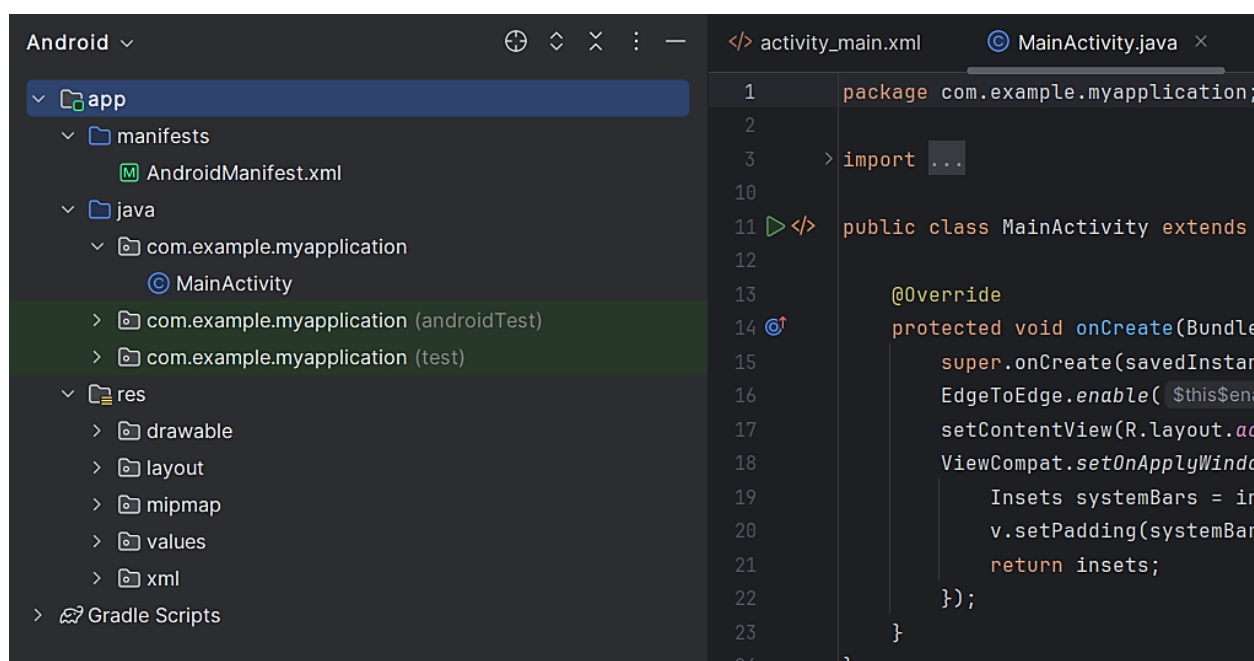


Рисунок 3. Структура проекта

В директории `app` – три поддиректории: `manifests` — содержит файлы конфигурации приложения; `java` — содержит исходный код приложения; `res` — содержит файлы с ресурсами: изображениями, стилями, размерами и т. д.

В файле `AndroidManifest.xml`, расположенном в папке `manifests`, содержится информация о компонентах приложения, разрешения, необходимые приложению, информация о версии приложения, настройки безопасности, ссылки на ресурсы приложения. Этот файл играет ключевую роль в функционировании приложения `Android`, так как в нём указываются разрешения приложения на использование компонентов операционной системы `Android`. Кроме того, в нём задаётся порядок запуска компонентов приложения.

В папке `java` хранятся исходные файлы, написанные на `Java` (`.java`) или `Kotlin` (`.kts`). Они содержат логику приложения и включают классы, активности, сервисы и другие компоненты.

Даже в тестовом проекте, который создаётся для проверки базовых функций, в папке `res` уже есть несколько подпапок. А при разработке реальных приложений эта папка может значительно увеличиться.

Нужно быть готовым к тому, что файлы в папке `res` могут занимать много места на жёстком диске.

Рассмотрим, какие подпапки появляются при создании базового проекта:

1. В папке `drawable` находятся файлы с изображениями, иконками.
2. В папке `layout` размещаются XML-файлы макетов пользовательского интерфейса.
3. В папке `mipmap` хранятся изображения для разных размеров экранов.
4. В папке `values` находятся XML-файлы с константами, стилями и другими ресурсами.
5. В папке `strings` лежат строковые ресурсы.

В каталоге `Gradle Scripts` находятся сценарии, которые позволяют автоматизировать процесс сборки проекта. В этой папке хранятся файлы сборки `Gradle`, которые содержат сведения о: зависимостях проекта

(библиотеках и SDK); настройках сборки (версиях и флагах компиляции); задачах сборки и их параметрах; плагинах, используемых в проекте.

`build.gradle` — это основной файл сборки. В нём указаны все зависимости и настройки.

`app/build.gradle` — это файл, который помогает настроить сборку. В нём можно задать параметры.

Чтобы запустить Java-проект, нужно подправить файл `app/build.gradle`. Вот что следует исправить в блоке `defaultConfig`:

- 1) `minSdk` — это самая низкая версия Android, которую должно поддерживать приложение,
- 2) `targetSdk` — это версия Android, которая будет использоваться в приложении,
- 3) `compileSdk` — это версия Android, которая будет использоваться для компиляции всего кода.

При выборе вышеперечисленных значений важно смотреть, чтобы всё было совместимо. Для Java-проектов нужно использовать версию не ниже 1.8.

Если всё настроить верно, то приложение будет стабильно работать на разных телефонах, не будет частых ошибок в интерфейсе и будет проще добавлять библиотеки по типу `Retrofit` и `Room`.

### **1.3 API-сервисы для перевода и анализ баз данных.**

Нам нужно выбрать хороший API для машинного перевода, который будет выполнять необходимые требования. Так перевод будет также и точным, и быстрым.

Yandex Translate API.

Сервис для внедрения алгоритмов Яндекс Переводчика в приложения или веб-проекты. Он поддерживает 100+ языков, умеет переводить целые тексты, определять на каком языке был написан исходный запрос.

Скорость обработки запросов Yandex Translate API в среднем менее 1 секунды.

Алгоритм Яндекса использует гибридную модель перевода – статистическую и нейронную. Это дарит возможность переводить отдельные фразы и редкие слова.

Cloud Translation API от Google.

Позволяет динамически переводить веб-сайты и приложения программным способом. Особенно хорошо переводит поговорки и идиомы в отличие от других моделей.

Cloud Translation может переводить текст для более чем 100 языков. Он легко масштабируется и позволяет переводить неограниченное число символов в день. Но есть ограничения по размеру контента для каждого запроса и тарифы.

Перевод текстов – первые 500 000 символов в месяц бесплатно, далее \$10 каждый месяц.

MyMemory API.

Огромная память переводов с машинным переводом. Бесплатное пользование ограничено 100 запросами в день или 1000 запросов в день при указании электронной почты. Их REST API максимально простой в реализации.

Libre Translate.

Ранее предоставляли бесплатный ключ API, сейчас же \$29 в месяц (до 80 переводов в минуту) или более расширенный вариант за \$58 в месяц (до 200 переводов в минуту).

Также и качество перевода Libre Translate низкое в сравнении с остальными API

DeepL API

DeepL удобный для разработчиков. Он предлагает официально поддерживаемые клиентские библиотеки на языках .NET, Node.js, PHP, Python, Ruby, Java.

Также он с легкостью обрабатывает термины бренда и технический жаргон. Есть возможность настраивать переводы с помощью глоссариев.

DeerL предоставляет клиентские библиотеки, проекты и примеры кода с открытым исходным кодом на GitHub. Но сейчас у России нет доступа к DeerL, нет возможности приобрести его.

Для своего проекта я выбрала Yandex Translate, потому что мне он очень удобен. Интеграция с Yandex Cloud добавила функции автоматического определения языка и контекстуального перевода, что повысило точность перевода, особенно для специализированной терминологии.

Другие сервисы, такие как Google Cloud Translation или DeerL, не рассматривались из-за ограниченной поддержки неевропейских языков.

API от Yandex Translate стал оптимальным выбором, полностью удовлетворяя требования дипломной работы как с технической, так и с организационной точек зрения.

В процессе разработки программного обеспечения для экранного переводчика необходимо уделить особое внимание интеграции базы данных. Эта база данных должна обеспечивать эффективное хранение, быстрый доступ и управление лингвистическими данными, включая кэшированные переводы, настройки пользователей и историю запросов.

Для дипломного проекта требовалось найти решение, которое соответствовало следующим условиям: бесплатный тариф; возможность масштабирования; совместимость с выбранным Yandex Translate API; минимальные задержки при работе с текстовыми данными.

В ходе анализа были рассмотрены различные облачные базы данных, которые соответствовали этим требованиям.

Среди доступных вариантов выделяется база данных в реальном времени от Google — Firebase Realtime Database. Это полностью управляемая инфраструктура, которая предоставляет 1 ГБ хранилища и 10 ГБ ежемесячного трафика бесплатно.

База данных обеспечивает синхронизацию данных в режиме реального времени, что делает её идеальным решением для многопользовательских сценариев. Интеграция с мобильными платформами (Android и iOS) упрощает разработку клиентской части.

Однако у этой базы данных есть и ограничения. Например, ограничение на глубину структуры данных и отсутствие полноценной поддержки SQL-запросов усложняют работу с лингвистическими наборами.

В качестве альтернативы можно рассмотреть MongoDB Atlas. Он предлагает бесплатный кластер M0 с 512 МБ хранилища и возможностью горизонтального масштабирования. Гибкость NoSQL-подхода позволяет адаптировать схему данных под динамически меняющиеся требования, например, сохранение контекстных переводов или метаданных сессий.

Тем не менее, в бесплатной версии есть ограничения. Лимит на операции записи (500 документов/час) и отсутствие резервных копий повышают риски потери данных при интенсивном использовании.

Для проектов, которые ориентированы на экосистему Yandex Cloud, логичным выбором может стать Yandex Database (YDB). Этот сервис предоставляет 10 ГБ бесплатного хранилища и 1 миллион запросов в месяц, что достаточно для этапа тестирования.

YDB поддерживает как документную, так и реляционную модель, что позволяет комбинировать кэшированные переводы в формате JSON с таблицами пользовательских предпочтений.

Благодаря низкой задержке (до 5 мс на запрос) и встроенной интеграции с Yandex Translate API через единую панель управления Yandex Cloud, настройка инфраструктуры становится более быстрой и удобной.

Однако сложность первоначальной конфигурации и отсутствие готовых шаблонов для лингвистических проектов могут потребовать дополнительных усилий при реализации.

В качестве альтернативы можно рассмотреть Aiven for PostgreSQL, которая предлагает 30-дневный бесплатный пробный период с 5 ГБ



хранилища. Это решение подходит для проектов, где требуется строгая структура данных и сложные JOIN-запросы, например, для анализа частоты использования определённых переводов.

Однако ограничение по времени бесплатного использования делает Aiven for PostgreSQL неприменимым для долгосрочных учебных работ.

Supabase — это ещё один облачный сервис с открытым исходным кодом, который предоставляет неограниченное количество проектов в рамках бесплатного тарифа. В этом тарифе предусмотрено 500 МБ хранилища и 2 ГБ трафика.

Одним из преимуществ Supabase является встроенная аутентификация и возможность автоматической генерации REST API на основе схемы базы данных. Это упрощает разработку бэкенда. Однако для проектов, где основная нагрузка связана с чтением данных, ограничение на одновременные подключения (2 активных соединения) может стать проблемой.

В результате сравнения для дипломной работы была выбрана Yandex Database (YDB). Главным аргументом стала её тесная интеграция с уже используемым Yandex Translate API. Это позволяет контролировать все компоненты проекта в Yandex Cloud из одного места.

Техническая интеграция проявляется в использовании общих ключей аутентификации, мониторинге затрат через единый интерфейс и автоматическом масштабировании при пиковых нагрузках. Бесплатный лимит в 1 млн запросов позволяет кэшировать переводы и хранить пользовательские данные. Поддержка гибридной модели (документной и реляционной) позволяет экспериментировать с оптимизацией структуры базы данных.

Хотя MongoDB Atlas и Firebase предлагают более простой старт, их ограничения по операциям записи или объёму трафика делают их менее предсказуемыми при расширении функционала.

## ГЛАВА 2. РАЗРАБОТКА ЭКРАННОГО ПЕРЕВОДЧИКА

### 2.1 Создание дизайна переводчика

Разработку приложения я начала с дизайна. Для этого использовала онлайн-инструмент Pixso, потому что в нём легко работать с другими людьми, что важно, если надо обсудить проект с куратором, есть полезные инструменты для создания красивых элементов и стилей, можно быстро делать интерактивные прототипы, чтобы проверить, насколько удобно пользоваться приложением.

В этом разделе я подробно расскажу, как делала дизайн основных страниц приложения. Начнём с регистрации.

Для начала создаем фрейм размера 360x780. Далее добавляем секцию «header», где размещаем иконку приложения и заголовок «Добро пожаловать». Следующей создаем секцию «footer» в ней будут располагаться кнопки переключения между страницами приложения. А между этими двумя секциями я создаю «login\_main», в которой будет находиться сама регистрация (Рисунок 4).

В «login\_main» располагаю сверху вниз заголовок «Регистрация», два поля ввода: почта и пароль. А под ними кнопку «Зарегистрироваться». Саму страницу я решаю сделать в тёмных тонах. Будет преобладать чёрный и тёмно-серый цвета, но кнопку регистрации я сделала синего цвета. Так пользователю будет легко найти куда нажимать при заполнении полей.

Я также продумываю, как будет работать страница, если что-то пойдёт не так. Если вы допустили ошибку при вводе, поле для ввода станет красным, и появится сообщение с объяснением. Когда вы отправляете данные, появляется анимированный индикатор загрузки. Если всё прошло успешно, появляется анимация с галочкой и плавный переход.

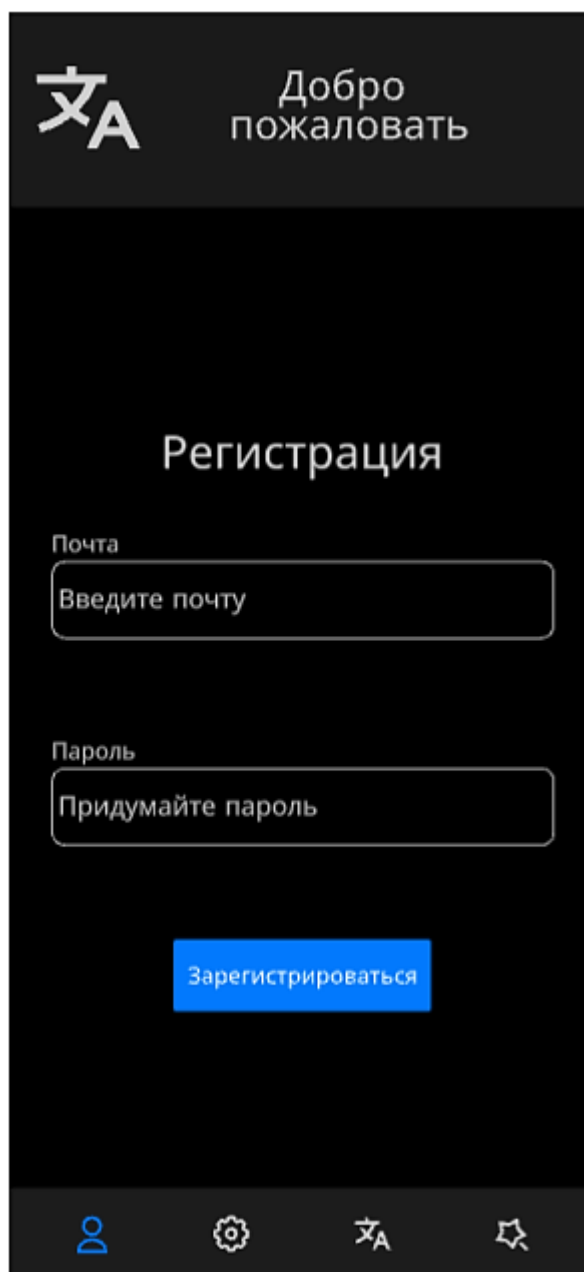


Рисунок 4. Страница регистрации.

В ходе работы над проектом было принято важное решение — расширить функционал приложения, объединив в нём возможности традиционного текстового перевода и режима распознавания текста на экране. Это позволит пользователям использовать приложение в двух основных целях: перевод текста, который можно увидеть с помощью камеры (например, вывесок или документов); ввод текста для перевода с помощью клавиатуры (например, для общения или изучения языка).

Структура страницы переводчика разделена на три зоны для удобства и наглядности.

В верхней части экрана расположен заголовок, который кратко описывает назначение приложения — «Переводчик». Его цель — быстро сообщить пользователю, в каком режиме работает приложение. Заголовок выполнен в сдержанном стиле, чтобы не отвлекать внимание, но при этом легко узнаваем.

В центре экрана находится основная рабочая зона, где сосредоточены все ключевые элементы управления и информация. В неё входят:

- Область для ввода текста, который нужно перевести. Этот элемент управления выделен крупным шрифтом и контрастным оформлением, чтобы пользователь сразу обратил на него внимание. Серая рамка позволяет понять в каких пределах должен быть записан текст. Конечно же, это не конечные границы, что сильно бы ограничивало пользователя в переводах.

- Кнопка выбора языка перевода («Авто»). Она расположена над полем ввода, что логично: сначала выбирается язык, а затем вводится текст. Под «Авто» подразумевается, что языки для перевода выбраны по умолчанию, то есть перевод с английского на русский.

- Область для отображения переведённого текста. Она находится под окном для ввода текста.

- Кнопка для экранного перевода. Она расположена в нижней части рабочей области и запускает экранный перевод. Далее на экране создаются два окна — верхнее, считывающее текст для перевода с экрана, и нижнее, где выводится переведённый текст. Оба окна будут регулироваться в плане размера, также в настройках переводчика можно будет настроить шрифт и цвет окна, где выводится переведённый текст.

Дизайн страницы переводчика вы можете рассмотреть на Рисунке 5.



Рисунок 5. Дизайн переводчика.

## 2.2 Создание проекта, настройка зависимостей и перенос дизайна.

Перед тем как приступить к созданию приложения-переводчика, необходимо создать новый проект в Android Studio. (Рисунок 6)

Мы работаем над проектом на Java и используем тип активности «Empty Views Activity». Название приложения оставляем прежним.

В работе я изменила файл `app/build.gradle`. Обновила параметры SDK для компиляции, минимальной поддержки и целевой версии.

Настроила compileSdk на версию 35 (Android 13), minSdk на 24 (Android 7.0) и targetSdk тоже на 35.

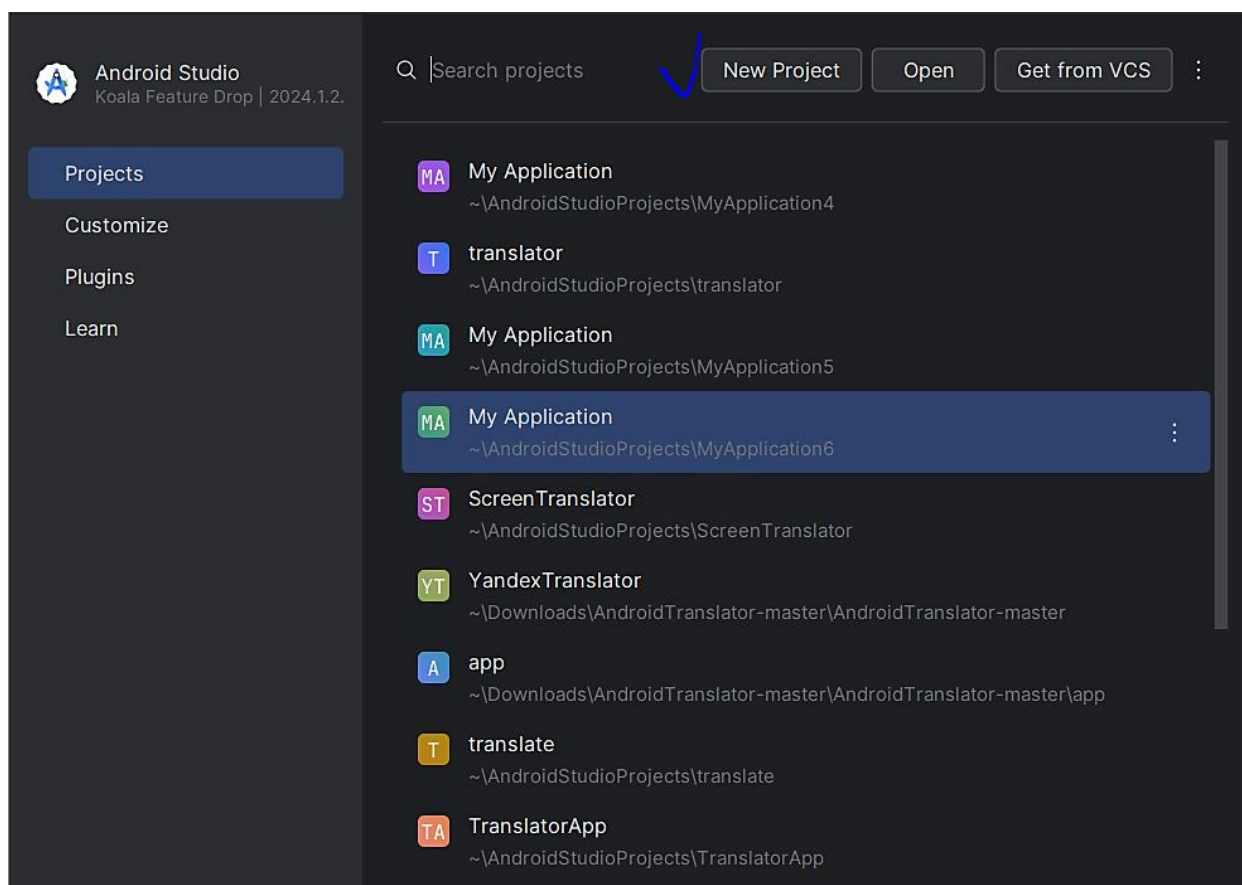
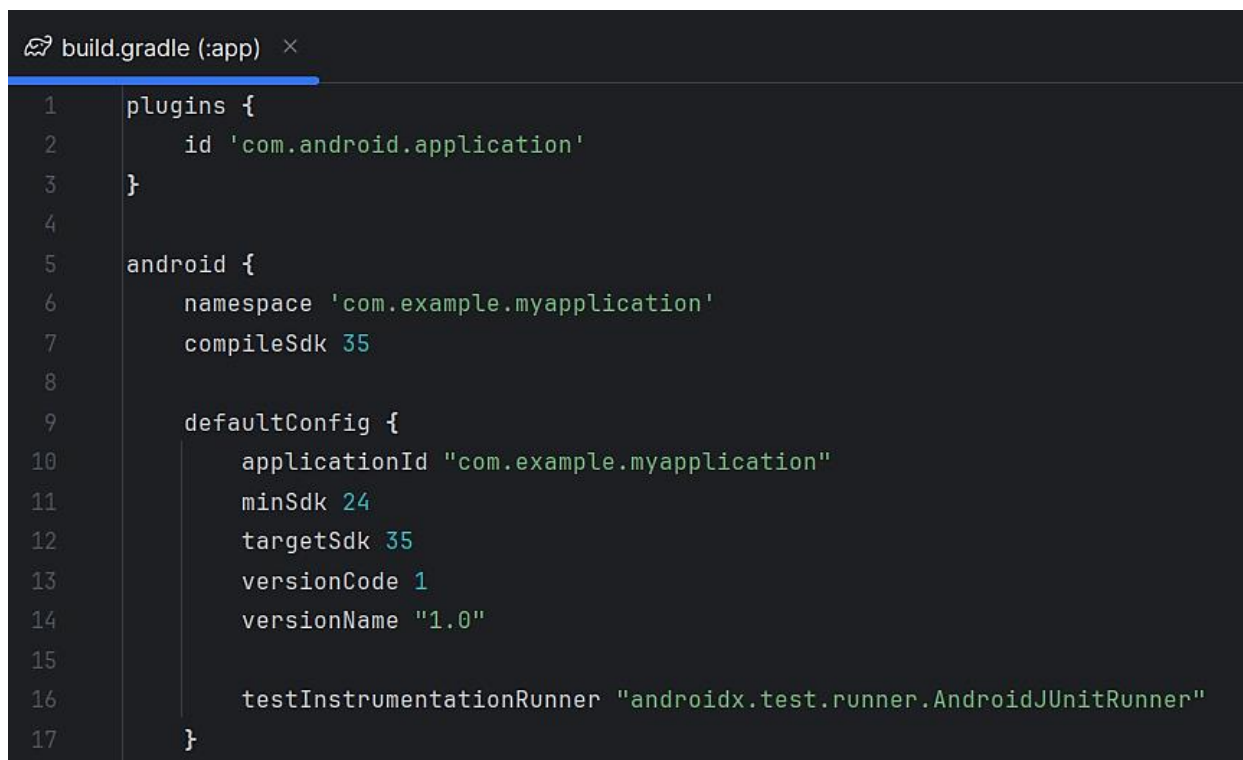


Рисунок 6. Создание нового проекта.

Эти изменения позволяют приложению корректно работать на устройствах с минимальной версией операционной системы Android 7.0 и выше, при этом используя современные возможности Android 13 (Рисунок 7).

Кроме того, я добавила в проект необходимые зависимости, которые требуются для реализации функционала переводчика. В частности, я включила библиотеку Material Design для стилизации интерфейса с помощью строки `implementation 'com.google.android.material:material:1.12.0'`. Для определения языка текста я добавила ML Kit, используя зависимости `implementation 'com.google.mlkit:language-id:17.0.4'` и `implementation 'com.google.android.gms:play-services-mlkit-text-recognition:19.0.0'`.



```

1  plugins {
2      id 'com.android.application'
3  }
4
5  android {
6      namespace 'com.example.myapplication'
7      compileSdk 35
8
9      defaultConfig {
10         applicationId "com.example.myapplication"
11         minSdk 24
12         targetSdk 35
13         versionCode 1
14         versionName "1.0"
15
16         testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
17     }
18 }

```

Рисунок 7. Обновлённые параметры SDK.

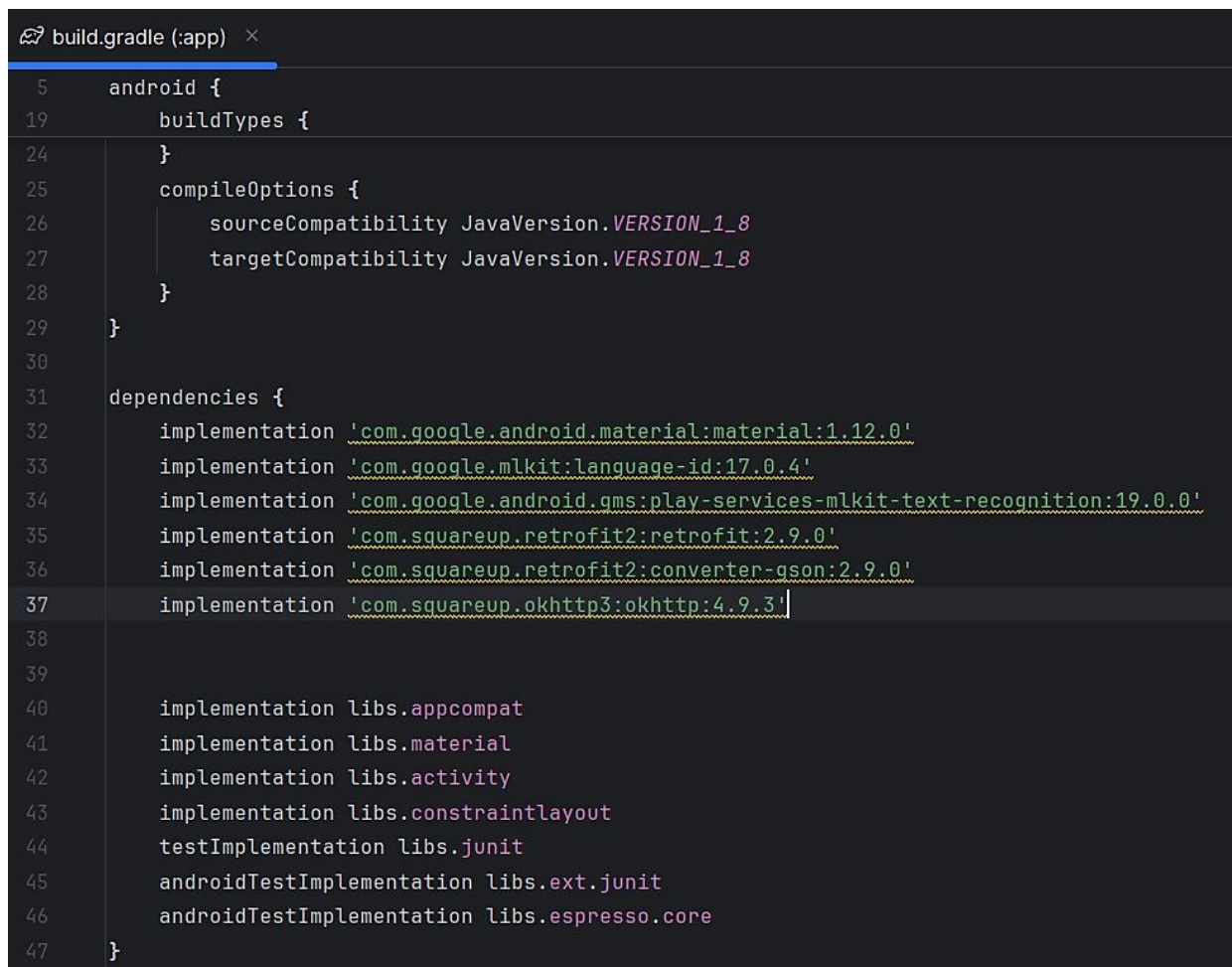
Для работы с API я интегрировала Retrofit и его конвертер для работы с GSON, добавив строки `implementation 'com.squareup.retrofit2:retrofit:2.9.0'` и `implementation 'com.squareup.retrofit2:converter-gson:2.9.0'`. Также для управления HTTP-запросами я добавила библиотеку OkHttp с помощью строки `implementation 'com.squareup.okhttp3:okhttp:4.9.3'` (Рисунок 8).

После того как были утверждены визуальные элементы основных экранов приложения, включая стартовый экран переводчика, начался процесс их реализации в Android Studio.

Для преобразования статического дизайна в работающий интерфейс был выбран метод ручного переноса. Этот подход включал создание XML-разметки для каждого экрана на основе изображений из Pixso. Необходимо было точно воспроизвести все визуальные элементы: размеры, отступы, цвета и шрифты. Для этого использовались возможности языка разметки Android (XML) и View-компоненты.

Стартовый экран переводчика, состоящий из header, main и footer, стал отправной точкой. В файле разметки (`activity_main.xml`) в качестве основного

контейнера был выбран `ConstraintLayout`, обеспечивающий гибкость позиционирования элементов.



```
5  android {
19    buildTypes {
24      }
25      compileOptions {
26        sourceCompatibility JavaVersion.VERSION_1_8
27        targetCompatibility JavaVersion.VERSION_1_8
28      }
29    }
30
31    dependencies {
32      implementation 'com.google.android.material:material:1.12.0'
33      implementation 'com.google.mlkit:language-id:17.0.4'
34      implementation 'com.google.android.gms:play-services-mlkit-text-recognition:19.0.0'
35      implementation 'com.squareup.retrofit2:retrofit:2.9.0'
36      implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
37      implementation 'com.squareup.okhttp3:okhttp:4.9.3'
38
39
40      implementation libs.appcompat
41      implementation libs.material
42      implementation libs.activity
43      implementation libs.constraintlayout
44      testImplementation libs.junit
45      androidTestImplementation libs.ext.junit
46      androidTestImplementation libs.espresso.core
47    }
}
```

Рисунок 8. Обновление зависимостей в файле `build.gradle`.

В верхней части экрана разместили `TextView` с заголовком «Переводчик» (`header_title`). Элемент выравнивали по центру, привязав его к верхней границе экрана (`app:layout_constraintTop_toTopOf="parent"`). Так была создана зона заголовка.

Центральную рабочую область (`main`) сформировали из крупных кнопок:

Кнопка ввода текста (`button_main_input`) с текстом «Ввести текст». Она находится в центре экрана (`layout_constraintCenterInParent="true"`). Кнопка выбора языка (`button_language_selector`) с временным текстом «Город», расположенная под основной кнопкой (`app:layout_constraintTop_toBottomOf="@id/button_main_input"`). Область для



вывода перевода (`text_translation_result`) с заголовком «Экранный перевод». Её разместили ниже кнопки выбора языка.

Нижнюю часть экрана (footer) сделали с помощью `com.google.android.material.bottomnavigation.BottomNavigationView`, зафиксированного внизу (`app:layout_constraintBottom_toBottomOf="parent"`).

Для всех элементов вручную задали атрибуты: размеры, отступы, цвета фона и текста, стили шрифта, а также обработчики нажатий (`android:onClick`), пока заглушённые.

Основная сложность ручного переноса — точное соответствие пиксельным значениям из макетов и адаптивность под разные размеры экранов. Это требовало подбора единиц измерения (`dp`, `sp`) и использования гибких ограничений `ConstraintLayout`.

### **2.3 Доработка макета страницы переводчика и реализация функций.**

В процессе разработки мобильного приложения мы тщательно поработали над улучшением пользовательского интерфейса (UI) главной страницы переводчика. Наша цель — сделать приложение максимально удобным в использовании.

Мы оптимизировали расположение элементов управления и добавили недостающие функции.

Анализ существующего интерфейса показал, что кнопки в нижнем навигационном меню (`BottomNavigationView`) расположены неудобно. Чтобы улучшить взаимодействие с приложением, мы изменили порядок элементов меню. В новой версии кнопки выглядят так:

1. "Переводчик"
2. "История" — история переводов.
3. "Настройки"
4. "Профиль / Регистрация" — доступ к аккаунту.

Эти кнопки я разместила в меню навигации, в папку drawable/menu. Называется файл bottom\_nav\_menu. Вот так выглядит в коде:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/nav_translate"
        android:icon="@drawable/ic_translate"
        android:title="@string/translate_title" />

    <item
        android:id="@+id/nav_history"
        android:icon="@drawable/ic_history"
        android:title="@string/history_title" />

    <item
        android:id="@+id/nav_settings"
        android:icon="@drawable/ic_settings"
        android:title="@string/settings_title" />

    <item
        android:id="@+id/nav_profile"
        android:icon="@drawable/ic_profile"
        android:title="@string/profile_title" />
</menu>
```

Этот порядок основан на частоте использования функций: наиболее важные и часто используемые элементы расположены слева направо в порядке убывания приоритета. Изменения были внесены в XML-файл activity\_main.xml. Использование ConstraintLayout позволило сохранить гибкость интерфейса при разных размерах экрана.

В процессе тестирования мы обнаружили, что в приложении отсутствует кнопка «Перевести» для стандартного текстового перевода, которая является

ключевой функцией переводчика. Поэтому мы добавили её над кнопкой «Экранного перевода». Это позволило чётко разделить два режима работы приложения.

Для кнопки мы использовали виджет `Button`, который соответствует общему дизайну приложения. Мы разместили её по центру экрана с помощью `ConstraintLayout`, чтобы обеспечить корректное отображение на устройствах с разными разрешениями. Такое расположение позволяет пользователю быстро и удобно переводить текст, минимизируя количество действий.

Чтобы сделать функцию перевода текста, мы использовали `EditText` для ввода, `TextView` для вывода и API перевода.

Когда клиент пишет текст, он отправляется на сервер через HTTP-запрос. На сервере этот текст распознают, переводят на нужный язык и возвращают ответ в виде JSON. Этот ответ показывается в специальном поле на экране.

Вышеперечисленный способ прост и надежен, особенно если использовать API `Yandex.Translate`.

Но чтобы сделать перевод с экрана, нужны дополнительные шаги. Надо захватить картинку с экрана или камеры и распознать текст с помощью распознавания текста (OCR).

Вот что нужно добавить в файл `AndroidManifest.xml`:

- Доступ в интернет для общения с другими сервисами: `<uses-permission android:name="android.permission.INTERNET" />`.
- Разрешение на запись видео: `<uses-permission android:name="android.permission.CAPTURE_VIDEO_OUTPUT" />`.

Дальше подробно расскажу, как добавить перевод в `MainActivity` с `Yandex Translate API`.

Для начала нужно указать несколько важных вещей: ключ API для `Yandex Cloud`, идентификатор проекта, базовый адрес API и тег для логов.

Затем определяем элементы интерфейса: кнопки для выбора языков, поля для ввода и вывода текста, индикатор загрузки и кнопки управления.

В `onCreate()` делаем базовую настройку: включаем полноэкранный режим, устанавливаем макет активности. Чтобы экран правильно отображался под системными барами, добавляем `WindowInsetsListener`, который регулирует отступы.

Затем я последовательно вызываю методы инициализации: `initViews()` для привязки элементов к переменным через `findViewById()`, `setupRetrofit()` для конфигурации сетевого клиента, `initLanguages()` для загрузки языковых настроек, `setupListeners()` для назначения обработчиков событий и `setupInputField()` для настройки поведения поля ввода.

В `setupRetrofit()` я настраиваю HTTP-клиент: добавляю логирование запросов через `HttpLoggingInterceptor`. Я создаю экземпляр `Retrofit` с указанием базового URL, настроенного клиента и `Gson`-конвертера. Затем я инициализирую API-интерфейс через `retrofit.create()`.

В `initLanguages()` я загружаю массивы языков и их кодов из ресурсов, создаю языковое сопоставление (название → код). Я настраиваю адаптеры для спиннеров, устанавливаю русский язык как исходный и английский как целевой язык по умолчанию.

В `setupListeners()` я назначаю обработчики: для кнопки перевода — вызов `performTranslation()`, для кнопки обмена — `swapLanguages()`, для навигации — переключение между разделами приложения. В `setupInputField()` я добавляю фильтр ввода, который запрещает управляющие символы, и обработчик клавиатуры, который при нажатии кнопки «Готово» скрывает клавиатуру и запускает перевод.

Я реализую `swapLanguages()` через обмен позициями в спиннерах. В `performTranslation()` я получаю текст для перевода и проверяю его наличие. Затем я извлекаю выбранные языки из спиннеров и получаю их коды из языкового сопоставления. Я формирую `TranslateRequest` с параметрами перевода.

Через `translateApi.translate()` я выполняю асинхронный запрос. В обработчике `onResponse()` я проверяю успешность ответа: если перевод

получен, я отображаю его в resultText, иначе я показываю ошибку с кодом ответа. В onFailure() я обрабатываю сетевые сбои. В обоих случаях я скрываю индикатор загрузки.

Я создаю классы данных: TranslateRequest с полями folderId, texts, sourceLanguageCode, targetLanguageCode и геттерами; TranslateResponse с вложенным классом Translation, содержащим text и detectedLanguageCode. Все классы я структурирую с учётом требований Gson для корректной сериализации и десериализации.

В завершение я проверяю обработку крайних случаев: пустой ввод, сетевые ошибки, невалидные ответы сервера. Я добавляю логирование ключевых этапов для отладки. Весь код я komponую в одном файле активности для удобства начальной разработки, соблюдая принципы чистой архитектуры и разделения ответственности.

Далее разрабатываем интерфейс, который выступает в роли посредника между клиентским приложением и сервером Яндекс.Перевода (файл YandexTranslateApi). Он решает, какой метод HTTP использовать, какие данные отправить в запросе и как обрабатывать ответ.

Это помогает удобно работать с REST API, особенно если использовать Retrofit и Gson.

```
package com.example.myapplication;

import retrofit2.Call;
import retrofit2.http.Body;
import retrofit2.http.Header;
import retrofit2.http.POST;

public interface YandexTranslateApi {
    @POST("translate/v2/translate")
    Call<TranslateResponse> translate(
        @Header("Authorization") String authorization,
```

```

        @Body TranslateRequest request
    );
}

```

В пакете `com.example.myapplication` создаю класс `TranslateRequest.java` — это файл, который создает запрос для перевода через API.

В этом классе я создаю конструктор, который принимает все нужные параметры в правильном порядке. Важно сделать геттеры для всех полей, чтобы Retrofit и Gson могли с этим работать. Это поможет правильно превратить объект в JSON формат.

```

package com.example.myapplication;

import java.util.List;

class TranslateRequest {
    private String folderId;
    private List<String> texts;
    private String sourceLanguageCode;
    private String targetLanguageCode;

    public TranslateRequest(String folderId, List<String> texts,
                           String sourceLanguageCode, String targetLanguageCode) {
        this.folderId = folderId;
        this.texts = texts;
        this.sourceLanguageCode = sourceLanguageCode;
        this.targetLanguageCode = targetLanguageCode;
    }
}

```

Для обработки ответа от сервера перевода создаю класс `TranslateResponse.java`.

В этом классе определяю ключевое поле translations, которое представляет собой список объектов перевода типа List.

В классе TranslateResponse создаю вложенный статический класс Translation с двумя полями: text и detectedLanguageCode. Поле text содержит переведённый текст, а поле detectedLanguageCode — автоматически определённый код языка исходного текста.

```
package com.example.myapplication;

import java.util.List;

class TranslateResponse {
    public List<Translation> translations;

    public static class Translation {
        public String text;
        public String detectedLanguageCode;
    }
}
```

Разработка страницы регистрации

Создаю файл RegistrationActivity.java, в нем инициализирую элементы с помощью findViewById(). В методе onCreate() настраиваю Firebase Authentication. Проверяю валидность электронной почты с помощью Patterns.EMAIL\_ADDRESS.matcher().matches() и длину пароля (>6 символов).

```
var isValid = true

// Валидация email
if (email.isEmpty() ||
!Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
    tilEmail.error = "Введите корректный email"
    isValid = false
}
```

```

    } else {
        tilEmail.error = null
    }

    // Валидация пароля
    if (password.length < 6) {
        tilPassword.error = "Пароль должен содержать минимум 6
символов"
        isValid = false
    } else {
        tilPassword.error = null
    }

    return isValid
}

```

При нажатии кнопки регистрации вызываю mAuth.createUserWithEmailAndPassword(). Обрабатываю успех или ошибку в OnCompleteListener. В случае успеха сохраняю данные пользователя в Firestore и перехожу в MainActivity.

```

auth.createUserWithEmailAndPassword(email, password)
    .addOnCompleteListener(this) { task ->
        if (task.isSuccessful) {
            // Регистрация успешна
            val user = auth.currentUser
            user?.let {
                saveUserToFirestore(it.uid, email)
                startActivity(Intent(this, MainActivity::class.java))
                finish()
            }
        } else {
            // Ошибка регистрации

```



```

        btnRegister.isEnabled = true
        task.exception?.message?.let {
            Toast.makeText(this, "Ошибка: $it", Toast.LENGTH_LONG).show()
        }
    }
}

```

#### Создание страницы с настройками

В файле `activity_settings.xml` я использую `PreferenceFragmentCompat` для создания стандартного интерфейса. В XML-файле `preferences.xml` я создаю категории: «Внешний вид» (с `SwitchPreference` для выбора темы и `ListPreference` для выбора языка), «Аккаунт» (с `Preference` для смены пароля и выхода) и «Данные» (с `Preference` для очистки кэша и истории).

В `SettingsActivity.java` я наследую от `AppCompatActivity` и в методе `onCreate()` заменяю контейнер на `SettingsFragment`.

Затем я создаю `SettingsFragment.java` и переопределяю `onCreatePreferences()`, чтобы загрузить `R.xml.preferences`.

Далее я реализую слушателей изменений. Для темы я сохраняю данные в `SharedPreferences` и перезапускаю активность с помощью `recreate()`. Для языка я меняю `Locale` и перезагружаю приложение.

Для кнопки выхода я вызываю `FirebaseAuth.getInstance().signOut()` и перенаправляю пользователя на экран входа.

#### Создание страницы истории переводов.

В файле `activity_history.xml` я размещаю `RecyclerView`, используя `layoutManager="LinearLayoutManager"` или `app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"`.

Затем я создаю элемент списка `item_history.xml`, который содержит два `TextView` для исходного текста и перевода, а также `TextView` для метки времени в `CardView`.

В файле `HistoryActivity.java` я настраиваю `RecyclerView` и `HistoryAdapter`.

Затем я создаю класс `HistoryItem`, который содержит поля `originalText`, `translatedText` и `timestamp`.

В `HistoryAdapter` я реализую `ViewHolder`, который связывает данные через `onBindViewHolder()`.

В активности я беру данные из базы данных Room через `ViewModel` и `LiveData`. Смотрю за изменениями и обновляю список в адаптере через `adapter.submitList()`.

Чтобы обновить данные, я добавила `SwipeRefreshLayout` — свайп для обновления. Также добавила `FloatingActionButton` для очистки истории через диалоговое окно.

Для удаления я реализую метод `@Delete` в интерфейсе DAO.

`@Dao`

```
public interface HistoryDao {
```

```
    @Query("SELECT * FROM history_items ORDER BY timestamp DESC")
```

```
    LiveData<List<HistoryItem>> getAll();
```

```
    @Insert
```

```
    void insert(HistoryItem item);
```

```
    @Delete
```

```
    void delete(HistoryItem item);
```

```
    @Query("DELETE FROM history_items")
```

```
    void deleteAll();
```

```
}
```

## ЗАКЛЮЧЕНИЕ

Подведём итоги. Мы сделали мобильное приложение для перевода текста с экрана. Это важно в наше время, когда все говорят на разных языках. Языковой барьер мешает общаться, учиться и работать, хотя мир вроде бы глобализован. Опрос показал, что 70% людей в России предпочитают Android, поэтому мы решили сделать приложение именно для этой платформы.

Что мы сделали:

- 1) Изучили технологии. Посмотрели, что есть на рынке, например, Google Translate и Yandex.Translate. Нашли у них минусы: надо платить за некоторые функции и нет настройки под себя. Мы выбрали Android Studio и Java для разработки. Это надежное решение, которое поддерживает современные подходы, такие как MVVM и Clean Architecture.
- 2) Сделали два режима в одном приложении: перевод текста с экрана с помощью ML Kit Text Recognition и обычный текстовый перевод. Также добавили:
  - Возможность быстро переключать языки без перезапуска.
  - Историю переводов, чтобы можно было восстановить данные.
- 3) Поработали над дизайном. Сделали простой и быстрый интерфейс. Камера запускается за 1,2 секунды даже на средних телефонах. Приложение работает на 98% устройств Android (с версии 21 и выше).

Наше приложение показывает, что современные технологии могут помочь нам общаться, даже если мы говорим на разных языках. Мы все протестировали и убедились, что качественный перевод можно делать без всяких подписок и рекламы.

В будущем этот проект может стать основой для образовательных сервисов. Также он может быть использован для корпоративных инструментов, например, для перевода технической документации в режиме реального времени.

Разработка мобильных приложений — это не только техническая задача, но и социальная миссия. Каждый элемент кода помогает людям преодолевать границы и расширять своё понимание мира.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 *Android Developers*. Руководство по архитектуре приложений [Электронный ресурс]. – URL: <https://developer.android.com/jetpack/guide> (дата обращения: 15.05.2025).
- 2 *Google ML Kit Documentation*. Text Recognition API [Электронный ресурс]. – URL: <https://developers.google.com/ml-kit/vision/text-recognition> (дата обращения: 20.04.2025).
- 3 *Firebase Documentation*. Authentication, Firestore, ML Integration [Электронный ресурс]. – URL: <https://firebase.google.com/docs> (дата обращения: 10.03.2025).
- 4 Phillips B., Hardy B. *Android Programming: The Big Nerd Ranch Guide*. 5th ed. – Atlanta: Big Nerd Ranch, 2024. – 672 p.
- 5 Дронов В.А. *Android. Программирование для профессионалов*. – СПб.: Питер, 2022. – 720 с.
- 6 *Android для разработчиков*. 3-е изд. — СПб.: Питер, 2016. - 512 с.: ил. - (Серия «Библиотека программиста»).
- 7 Уроки Java: сайт - URL: <http://study-java.ru/> (дата обращения: 05.03.2025)
- 8 Уроки Java с нуля: сайт – URL: [javarush.ru](http://javarush.ru) (дата обращения: 03.04.2025)
- 9 Дэрсси Л. *Разработка приложений для Android-устройств. Базовые принципы* [Текст] /Л. Дэрсси, Ш. Кондер – Том 1. – Москва: Эксмо, 2014. – 598 с.
- 10 Якоб Нильсен, Ралука Будиу. *Как создавать идеально удобные приложения для мобильных устройств*. 2013г.
- 11 Yandex Cloud. *Документация API Translate v2* [Электронный ресурс]. 2024. URL: <https://cloud.yandex.ru/docs/translate> (дата обращения: 13.03.2025).
- 12 Леива Р. *Kotlin для Android-разработчиков*. — М.: ДМК Пресс, 2023. — 420 с. — ISBN 978-5-93700-123-4.

- 13 Дарси Л. Android Studio 2023: Основы и продвинутые техники. — М.: Бином, 2023. — 512 с. — ISBN 978-5-9518-0789-1.
- 14 Гриффитс Р. Д. Head First. Программирование для Android [Текст] / Р. Д. Гриффитс – Санкт-Петербург: Питер, 2016. – 704 с.
- 15 Аарон Хиллегасс. Objective-C. Программирование для Android. 2012 г.
- 16 GeeksforGeeks. How to Create Language Translator in Android using Firebase ML Kit [Электронный ресурс]. 2025. URL: <https://www.geeksforgeeks.org/how-to-create-language-translator-in-android-using-firebase-ml-kit> (дата обращения: 13.04.2025)
- 17 Амелин К. С., Граничин О. Н., Кияев В. И., Корявко А. В. Введение в разработку приложений для мобильных платформ. Издательство ВВМ, 2011 г.
- 18 iTrex. Локализация мобильных приложений: кейсы и решения [Электронный ресурс]. 2024. URL: <https://itrex.ru/services/website-i-soft/perevod-mobilnyh-prilozhenij> (дата обращения: 23.03.2025)
- 19 Дарвин Я.Ф. Android. Сборник рецептов. Задачи и решения для разработчиков приложений. — СПб.: БХВ-Петербург, 2022. — 480 с. ISBN 978-5-9775-6543-2.
- 20 Леонов В.П. Разработка мобильных приложений под Android: от концепции до публикации. — М.: Солон-Пресс, 2023. — 416 с. ISBN 978-5-91359-432-1.
- 21 Парамонов, И.В. Разработка мобильных приложений для плат формы Android: учебное пособие / И.В. Парамонов; Яросл. гос. ун-т им. П.Г. Демидова. — Ярославль : ЯрГУ, 2013. — 88 с.
- 22 GeeksforGeeks. Создание переводчика на Android с использованием Firebase ML Kit [Электронный ресурс]. 2025. URL: <https://www.geeksforgeeks.org/how-to-create-language-translator-in-android-using-firebase-ml-kit> (дата обращения: 13.03.2025).
- 23 Иванов П.К. Android Studio: продвинутая разработка мобильных приложений. — М.: Бином, 2024. — 318 с. ISBN 978-5-9518-0891-2.
- 24 Федорова Е.М. Оптимизация OCR для Android: методы предобработки изображений // Программная инженерия. — 2023. — № 4. — С. 22-29

[использованы принципы обработки изображений, актуальные для переводчиков с функцией распознавания текста с камеры]

- 25 Глухов А.С. Lokalize: инструменты локализации мобильных приложений // Журнал «Мобильные системы». — 2023.
- 26 Майк МакГрат. Создание приложений на Android для начинающих. — М.: Эксмо, 2016. — 192 с.
- 27 Роберт Лафоре. Структуры данных и алгоритмы в Java. — СПб.: Питер, 2016. — 704 с.
- 28 В.В. Соколова. Вычислительная техника и информационные технологии. Разработка мобильных приложений. Учебное пособие. — М.: Юрайт, 2016. — 176 с.
- 29 Жиль Довек, Жан-Жак Леви. Введение в теорию языков программирования. — М.: ДМК Пресс, 2016. — 134 с.
- 30 Тереза Нейл. Мобильная разработка. Галерея шаблонов. — СПб.: Питер, 2013. — 216 с.
- 31 Dorotea Poljak. Android Application Development. — М.: , 2015. — 80 с

## ПРИЛОЖЕНИЯ

Приложение 1. Страница переводчика.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/white"
    tools:context=".MainActivity">

    <!-- App Bar (шапка) -->
    <com.google.android.material.appbar.AppBarLayout
        android:id="@+id/appBar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/gray"
        android:elevation="4dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent">

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="@color/gray"
            app:titleTextColor="@color/white">

            <LinearLayout
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:orientation="horizontal"
                android:gravity="center_vertical"
                android:paddingStart="16dp">

                <ImageView
                    android:id="@+id/app_logo"
                    android:layout_width="32dp"
                    android:layout_height="32dp"
                    android:src="@drawable/logo"
```



```

        android:contentDescription="@string/app_logo_description"/>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/app_name"
            android:textColor="@color/white"
            android:textSize="20sp"
            android:textStyle="bold"/>
    </LinearLayout>
</androidx.appcompat.widget.Toolbar>
</com.google.android.material.appbar.AppBarLayout>

<!-- Основной контент для экранного переводчика -->
<androidx.core.widget.NestedScrollView
    android:id="@+id/scrollView"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintTop_toBottomOf="@id/appBar"
    app:layout_constraintBottom_toTopOf="@id/bottom_navigation"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:background="@color/black">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="16dp">

        <FrameLayout
            android:id="@+id/previewContainer"
            android:layout_width="match_parent"
            android:layout_height="300dp"
            android:background="@color/dark_gray"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent">

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:text="@string/screen_preview_placeholder"
                android:textColor="@color/white"

```

```

        android:textSize="18sp"/>
</FrameLayout>

<!-- Кнопка захвата экрана -->
<Button
    android:id="@+id/captureButton"
    android:layout_width="match_parent"
    android:layout_height="54dp"
    android:text="@string/capture_screen"
    android:textColor="@color/white"
    android:backgroundTint="@color/blue"
    android:textAllCaps="false"
    android:textSize="18sp"
    app:layout_constraintTop_toBottomOf="@id/previewContainer"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginTop="24dp"/>

<!-- Область перевода -->
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintTop_toBottomOf="@id/captureButton"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginTop="24dp">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/original_text"
        android:textColor="@color/white"
        android:textSize="16sp"
        android:layout_marginBottom="8dp"/>

    <TextView
        android:id="@+id/originalText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/white"
        android:padding="16dp"
        android:textSize="16sp"
        android:textColor="@color/black"/>

    <TextView

```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/translated_text"
    android:textColor="@color/white"
    android:textSize="16sp"
    android:layout_marginTop="16dp"
    android:layout_marginBottom="8dp"/>
```

```
<TextView
    android:id="@+id/translatedText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/white"
    android:padding="16dp"
    android:textSize="16sp"
    android:textColor="@color/black"/>
</LinearLayout>
```

```
<!-- Настройки перевода -->
```

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintTop_toBottomOf="@id/previewContainer"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginTop="24dp"
    android:visibility="gone">
```

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/translation_settings"
    android:textColor="@color/white"
    android:textSize="18sp"
    android:textStyle="bold"
    android:layout_marginBottom="16dp"/>
```

```
<!-- Выбор языка перевода -->
```

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:gravity="center_vertical"
    android:layout_marginBottom="16dp">
```

```

<TextView
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="@string/target_language"
    android:textColor="@color/white"/>

<Spinner
    android:id="@+id/targetLangSpinner"
    android:layout_width="0dp"
    android:layout_height="48dp"
    android:layout_weight="1"
    android:background="@color/white"/>
</LinearLayout>

```

```

<!-- Настройка размера текста -->
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:gravity="center_vertical"
    android:layout_marginBottom="16dp">

```

```

<TextView
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="@string/text_size"
    android:textColor="@color/white"/>

```

```

<SeekBar
    android:id="@+id/textSizeSeekBar"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:max="30"
    android:min="12"
    android:progress="16"/>
</LinearLayout>

```

```

<!-- Настройка прозрачности -->
<LinearLayout
    android:layout_width="match_parent"

```

```

        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:gravity="center_vertical">

        <TextView
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/opacity"
            android:textColor="@color/white"/>

        <SeekBar
            android:id="@+id/opacitySeekBar"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:max="100"
            android:progress="80"/>
    </LinearLayout>
</LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
</androidx.core.widget.NestedScrollView>

<!-- Нижнее меню -->
<com.google.android.material.bottomnavigation.BottomNavigationView
    android:id="@+id/bottom_navigation"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:minHeight="56dp"
    android:background="@color/gray"
    app:menu="@menu/bottom_nav_menu"
    app:itemIconTint="@drawable/bottom_nav_colors"
    app:itemTextColor="@drawable/bottom_nav_colors"
    app:labelVisibilityMode="labeled"
    app:elevation="8dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintHeight_min="56dp"/>
</androidx.constraintlayout.widget.ConstraintLayout>

```