

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА ТЕХНОЛОГИЙ ПРОГРАММИРОВАНИЯ

Направление: 09.03.03 – Прикладная информатика

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ИЗУЧЕНИЯ
ИНОСТРАННЫХ ЯЗЫКОВ

Работа завершена:

Студент 4 курса
группы 09-751

« 4 » июня 2021 г.  Галеева А.М.

Работа допущена к защите:

Научный руководитель

канд. физ.-мат. наук, доцент

« 4 » июня 2021 г.  Медведева О.А.

Заведующий кафедрой

канд. экон. наук, доцент

« 4 » июня 2021 г.  Вахитов Г.З.

Казань-2021

Содержание

Аннотация.....	3
Введение	5
1. Анализ существующих мобильных приложений и постановка задачи на разработку.....	8
1.1 Обзор и характеристика существующих мобильных приложений для изучения иностранных языков	8
1.2 Постановка задачи на разработку и выбор инструментов	16
2. Проектирование мобильного приложения для изучения иностранных языков	21
2.1 Проектирование базы данных	21
2.2 Проектирование пользовательского интерфейса	25
3. Разработка мобильного приложения.....	33
3.1 Разработка функционала приложения	33
3.2 Внедрение систем перевода и добавления слов	40
3.3 Оптимизация работы приложения на различных устройствах.....	45
4. Тестирование приложения.....	49
4.1 Тестирование функциональных элементов мобильного приложения	49
4.2 Тестирование алгоритма распознавания.....	51
Заключение	54
Список использованных источников	58
Приложения	

Аннотация

В работе рассматривается создание мобильного приложения для изучения иностранных языков. Изучение основывается на расширении словарного запаса путем использования системы на основе метода Лейтнера. Для выполнения задачи были рассмотрены существующие решения, применены в работе современные сервисы перевода и распознавания текста, а также интегрированы новые функции для добавления слов. Разработка приложения велась под операционную систему Android с использованием языка программирования Kotlin, среды разработки Android Studio, базы данных SQLite, сервиса перевода Google Cloud Translation, сервиса распознавания Google Mobile Vision.

Разработанное приложение соединяет в себе: удобный интерфейс с минималистичным дизайном интерфейса, удобную систему управления и навигации, интегрированный сервис перевода для быстрого перевода текста на сотни языковых пар внутри приложения, сервис по распознаванию текста с камеры устройства, функцию, позволяющую в один клик добавить скопированный текст из других приложений в нужный набор слов.

Ключевые слова: Android, мобильное приложение, Kotlin, SQLite, Google Cloud Translation, Google Mobile Vision.

Annotation

In this paper we consider the developing of mobile application for studying foreign languages. The study process is based on expanding vocabulary by using a system based on the Leitner method. To complete the task were taken into account existing solutions, applied modern translation and text recognition services and integrated new functions for adding words. The application was developed for the Android operating system using Kotlin programming language, Android Studio development environment, SQLite database, Google Cloud Translation service for translation and Google Mobile Vision text recognition service.

The developed application combines: a user-friendly interface with a minimalistic design, a convenient control and navigation system, an integrated translation service for quick text translation into hundreds of language pairs within the application, a service for text recognition from the device's camera, a function that allows you to add copied text from other applications to the desired set of words in one click.

Key words: Android, mobile application, Kotlin, SQLite, Google Cloud Translation, Google Mobile Vision.

Введение

В современном мире во многих сферах жизни информационные технологии оказали и оказывают большое влияние. Сейчас уже невозможно представить многие виды работ и досуга без них. Так и в обучении применяется большое количество нововведений, связанных с внедрением различных вспомогательных технологий, которые призваны повысить удобство, качество и скорость обучения.

Актуальность данной работы заключена в том, что обучение, связанное с расширением и удержанием активного словарного запаса, имеет важное значение для освоения нового языка и поддержке знаний об уже освоенных не только для личных нужд, но и для профессиональной деятельности. Во-первых, в наше время на многие рабочие позиции необходимо знание как минимум одного иностранного языка, а к кандидатам, желающим занимать более высокую должность, данные требования еще выше, и такие тенденции наблюдаются во многих сферах. Во-вторых, на любом этапе изучения языков требуется нарабатывать словарный запас, который будет служить фундаментом для последующего прогресса в изучении. В-третьих, наработка словарного запаса нужна не только для иностранных языков, но и для родного языка, так как большой запас помогает выражать свои мысли более креативно и дает возможность говорить о многих вещах без специальной подготовки.

Сам словарный запас - совокупность слов, которыми владеет человек. Различают два типа словарного запаса: активный и пассивный. Активный словарный запас включает слова, которые человек способен использовать в устной речи (особенно спонтанной) и письме, не задумываясь перед их применением. К пассивной части словарного запаса относятся слова, которые человек узнает при чтении или на слух, но не использует их в своей устной речи или письме.

Целью данной выпускной квалификационной работы является разработка мобильного приложения для расширения словарного запаса

иностранных языков и удобного закрепления новых слов и фраз, предназначенного для различных устройств на базе Android. Для достижения поставленной цели необходимо решить следующие задачи:

1. проанализировать аналогичные существующие приложения;
2. определить функционал приложения;
3. выбрать инструменты для разработки;
4. спроектировать интерфейс системы и систему хранения данных;
5. разработать мобильное приложение;
6. протестировать мобильное приложение.

Основные преимущества использования мобильного приложения для расширения активного словарного запаса при изучении иностранных языков заключается в следующем:

- большая мобильность – пользователь может организовать свой учебный процесс в удобном месте и в удобное время;
- обеспечение непрерывности обучения;
- более простое соблюдение принципов выбранной системы обучения;
- наличие полезных сервисов, помогающих в обучении.

Для разработки приложения под операционную систему Android была использована интегрированная среда разработки Android Studio. В качестве языка программирования был выбран язык Kotlin. Для хранения данных приложения была выбрана база данных SQLite. В качестве сервиса переводчика был выбран Google Cloud Translation, а в качестве сервиса распознавания Google Cloud Vision. Для тестирования приложения использовались библиотеки Espresso и JUnit.

Выпускная квалификационная работы состоит из введения, четырех глав, заключения, списка используемых источников и приложения.

В введении содержится информация об актуальности работы, цели и поставленных задачах.

Первая глава «Анализ существующих мобильных приложений и постановка задачи на разработку» посвящена разбору выбранной темы, существующих решений для расширения словарного запаса, а также постановке требований и выбору инструментов для реализации проекта.

Вторая глава «Проектирование мобильного приложения для изучения иностранных языков» содержит в себе описание проектирования важных составляющих мобильного приложения: базы данных и пользовательского интерфейса.

Третья глава «Разработка мобильного приложения» включает в себя описание процесса разработки мобильного приложения, интегрирование систем перевода и добавления слов. Внимание в нём уделяется не только корректности работы программного обеспечения, но и пользовательскому интерфейсу: его простоте и удобству использования.

В четвертой главе «Тестирование приложения» описывается процесс тестирования разработанного приложения. Тестирование производится как вручную, так и при помощи автоматизированных тестов.

В заключении описывается общий вывод о ходе проделанной работы. Список используемых источников содержит необходимую литературу и официальную документацию по теме ВКР. В приложении представлены фрагменты кода программы.

Вся работа над проектом проводилась с соблюдением норм чередования умственной и физической нагрузки.

В рамках выполнения ВКР была использована различная учебная литература, специализированные средства разработки программного обеспечения и их официальная документация с сайтов разработчиков, а также интернет-ресурсы и научно-практические статьи.

1. Анализ существующих мобильных приложений и постановка задачи на разработку

1.1 Обзор и характеристика существующих мобильных приложений для изучения иностранных языков

Согласно поставленным задачам, перед началом разработки мобильного приложения для изучения иностранных языков необходимо провести исследование функционала уже существующих на данный момент приложений – конкурентов. Приложения для обзора выбирались из категории «Образование» в магазине приложений Google Play, так как данный агрегатор является крупнейшим из позволяющих сторонним компаниям и разработчикам предлагать владельцам устройств на операционной системе Android различные приложения.

Перед началом выбора приложений для обзора опишем основную методику, на основе которой будет работать приложение.

Для изучения языка путем расширения словарного запаса при помощи использования мобильного приложения была выбрана система, основанная на системе Лейтнера [5, с. 23-34]. Данная система нашла широкое применение в различных сферах, где возможно применить использование флэш-карточек для эффективного запоминания и повторения. Метод был разработан немецким ученым и журналистом Себастьяном Лейтнером в 70-е годы XX века. Основная идея метода заключается в следующем:

1. составляется список вопросов с ответами;
2. далее вопросы и ответы переносятся на карточки. С одной стороны располагается вопрос, а с другой ответ не него;
3. пользователь случайным образом выбирает карточку из общего набора и пытается ответить на вопрос, который написан на ней;
4. для определения был ли дан верный ответ, пользователь переворачивает карточку на другую сторону.

5. в конце производится сортировка флеш-карточек по группам в зависимости от того, насколько хорошо пользователь усвоил информацию на каждой из них.

Данная методика чаще всего используется для запоминания новых слов, исторических дат и формул. Она является доступной каждому, не требуя больших затрат для применения, так как карточки можно сделать из подручных средств. Сам метод помогает воспроизвести эффект тестирования. Правильное применение данного метода показывает, что время хранения информации в долговременной памяти увеличивается, когда часть периода обучения посвящена извлечению информации посредством тестирования и мгновенной проверкой ответа. На данный момент существует ряд программ, которые используют этот метод.

В самом приложении предполагается реализация 3-х групп для карточек с разным периодом повторения слов в них. По данным научных исследований 3 группы – это наиболее оптимальное количество для флеш-карточек:

- в первую группу помещаются новые слова и слова, усвоенные плохо;
- во вторую группу помещаются слова, которые усвоены лучше;
- в третью группу помещаются слова, которые усвоены пользователем хорошо.

Если пользователь выбирает правильный ответ на слово из группы 1 и у слова набирается определенный процент правильных ответов, то карточка перемещается в группу 2. Перемещение карточек по другим группам происходит аналогично. Если пользователь допускает ошибку при выборе правильного значения на карточке из группы 2 или 3, то после достижения граничного процента карточка возвращается в группу 1. Повторный показ слов в программе зависит от количества набранных правильных ответов. Чем больше правильных ответов, тем реже слово появляется для показа. За каждый правильный ответ по переводу слова начисляется 1 балл, за неправильный – вычитается 2.

Для оценки работы приложений, подходящих под тематику разрабатываемого, был выбран список критериев, который отражает основной планируемый функционал разрабатываемого приложения, а именно:

1. возможность автоматического перевода слов на выбранный язык;
2. возможность добавления слов не только путем введения символов с клавиатуры, но и посредством использования других инструментов.
3. наличие возможности протестировать усвоение информации.

В магазине приложений Google Play одним из самых популярных приложений, использующих в своей основе систему Лейтнера, является приложение «Quizlet». Его создатели позиционируют приложение прежде всего как эффективный инструмент для изучения иностранных языков, но большинство пользователей приложения адаптировали систему изучения и под другие сферы, такие как: математика, химия, наука и многое другое.

Приложение «Quizlet» имеет большой функционал доступный для бесплатного использования. Также страница приложения в магазине имеет большое количество отзывов, и его средняя оценка составляет 4.6 по пятибалльному рейтингу. Однако, проанализировав отзывы за последние полгода было отмечено, что множество пользователей испытывают затруднения при работе с приложением, а именно:

1. периодические зависания;
2. потеря прогресса в изучении слов;
3. отсутствие возможности дублирования уже существующих карточек.
4. исчезновение добавленных слов.

Стоит отметить, что большинство ошибок создатели стараются исправить в кратчайшее время, выпуская обновления.

Приложение предоставляет возможность создания папок, учебных наборов из слов для различных нужд. Добавление новой информации может происходить как при помощи ввода с клавиатуры (см. Рисунок 1), так и при помощи добавления фотографий с камеры устройства или внутреннего

хранилища (см. Рисунок 2). Второй метод доступен лишь в платной подписке для приложения, плата за которую составляет 1490 руб./ год.



Рисунок 1. Экран добавления в набор

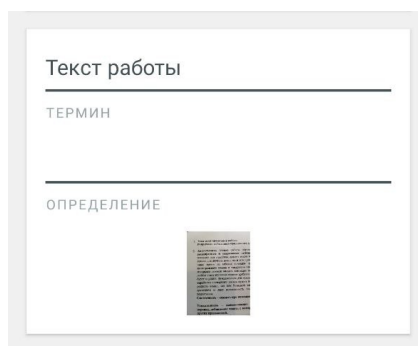


Рисунок 2. Пример добавления фотографии

В приложении присутствует возможность озвучки слов в флэш-карточках, также функция добавления карточек в избранную группу, которая отображается дополнительной вкладкой. Для изучения новых слов предлагается использовать различные модификации метода Лейтнера и другие методы (см. Рисунок 3). Прогресс в изучении отображается для каждого сета в процентах. Интерфейс приложения приятный и минималистичный.

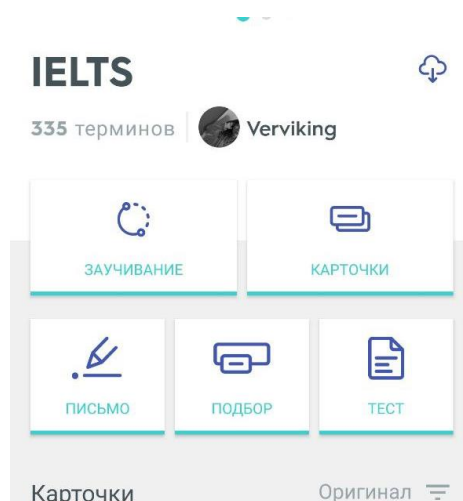


Рисунок 3. Методы изучения

Подводя итоги, к плюсам данного приложения можно отнести функционал, различные методы ввода новой информации для изучения, регулярную поддержку от создателей. Пользователи также выделяют функцию перевода слов и удобный интерфейс программы. Из минусов можно выделить отсутствие добавления фотографий в карточку и присутствие рекламы в бесплатном варианте приложения.

Второе приложение «Флэшкарты», выбранное для обзора, также как и упомянутый выше Quizlet, работает по системе Лейтнера. Функционал приложения также схож. Приложение имеет два основных экрана: главный экран с наборами слов (см. Рисунок 4) и экран с отображением карточек набора (см. Рисунок 5). В наборе карточки помимо редактирования можно добавить в избранную группу, прослушать произношение добавленных в нее слов. Добавление новых карточек происходит только посредством ввода текста с клавиатуры. Пользователю доступны различные методы проверки количества запомненных слов, такие как выбор правильного ответа из нескольких предложенных, самостоятельное определения степени запоминания слова (см. Рисунок 6). Также пользователь может получить отчет о прогрессе. Стоит отметить, что детальный отчет о прогрессе доступен только в платной версии, стоимость которой составляет 250 рублей. Данная сумма оплачивается единовременно и позволяет использовать функции приложения без

ограничения по времени. Также платная версия отключает рекламу и позволяет делать резервную копию данных.

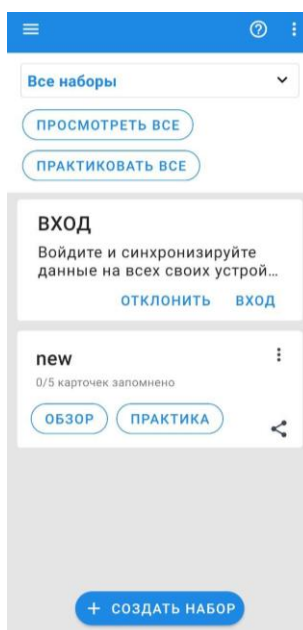


Рисунок 4. Главный экран «Флэшкарты»

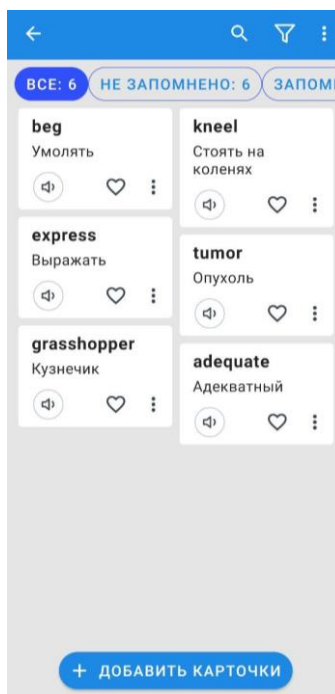


Рисунок 5. Экран с карточками набора



Рисунок 6. Выбор степени запоминания слова

К достоинствам данного приложения можно отнести понятный интерфейс, удобную шкалу разметки и методы проверки изучения слов. К минусам можно отнести отсутствие ввода новых слов при помощи других способов, а также отсутствие автоматического перевода.

Третье приложение, выбранное для обзора – «Flashcards». Из всех рассмотренных приложений данное имеет самый простой интерфейс и функционал. В приложении имеется основной экран со списком наборов и экран с добавлением карточек (см. Рисунок 7).

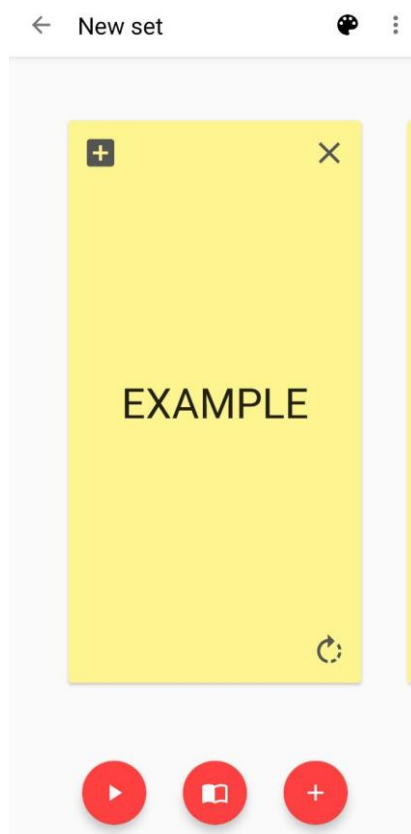


Рисунок 7. Экран просмотра и добавления карточек

В приложении можно также можно пройти упрощенную версию тестирования в виде вопроса «помню – не помню» к каждой карточке и просмотреть статистику обучения (см. Рисунок 8). Несмотря на простоту интерфейса, приложение оказалось самым удобным в использовании, так как не было перегружено функционалом. Однако здесь не используется сервис для автоматического перевода слов, что однозначно сокращает круг потенциальных пользователей приложения. Весь доступный функционал может быть использован в бесплатной версии пользователю, так как приложение не имеет платных функций.

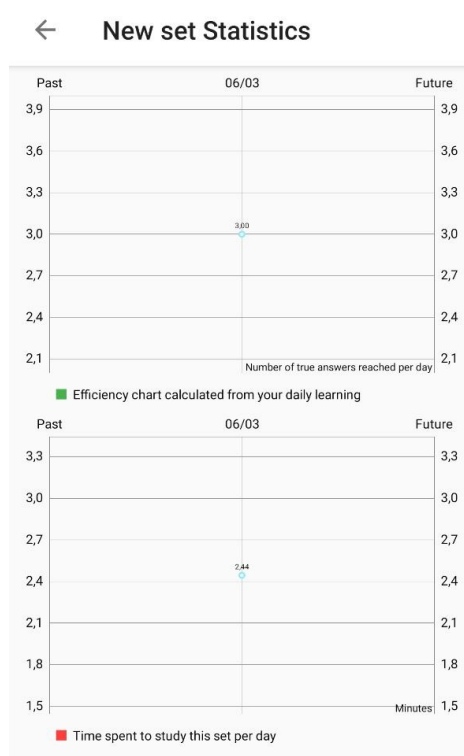


Рисунок 8. Статистика обучения

Все три рассмотренных приложения используют в основе своего интерфейса «плавающую» кнопку в углу экрана для добавления карточек или наборов. Данное решение является выгодным, так как такая кнопка обеспечивает удобный доступ к часто используемой функции добавления. Также было отмечено, что удобной реализацией является показ на главном экране приложения списка наборов, нажав на которые можно перейти непосредственно к содержимому.

При разработке приложения будут приняты во внимание и учтены результаты обзора всех приложений, что поможет разработать систему, лишенную недостатков других систем, а также имеющее свои функциональные особенности.

1.2 Постановка задачи на разработку и выбор инструментов

Разрабатываемое мобильное приложение предназначено для расширения словарного запаса иностранных языков и удобного закрепления новых слов и фраз. Для успешной разработки данного приложения необходимо определить целевую аудиторию, определить основные экраны для создаваемого интерфейса, а также перечень реализуемых функций, необходимых для продуктивной работы с программой.

Целевая аудитория приложения может включать в себя:

- студентов и школьников, изучающих иностранные языки в рамках учебного процесса;
- работников, которым знание иностранных языков необходимо для занимаемой должности;
- других людей, которые по тем или иным причинам (путешествия, интерес, расширение кругозора и другое) хотят изучить новый язык.

Для того, чтобы приложение помогало пользователям эффективно запоминать новые слова и фразы, тем самым расширяя их словарный запас, необходимо:

- правильно реализовать подход к изучению, используя в основе систему Лейтнера;
- реализовать удобное добавление слов в приложение как при помощи их ввода с клавиатуры, так и при помощи считывания с камеры устройства и копирования из других приложений;
- реализовать гибкую и удобную систему управления данными, в особенности дублирование карточек;

- реализовать возможность просмотра прогресса обучения;
- реализовать отправку регулярных оповещений для напоминания о необходимости повторений для поддержания оптимального графика обучения.

После разбора и анализа приложений – конкурентов было решено добавить в приложение 4 основных экрана:

- экран со списком имеющихся наборов слов;
- экран с функционалом для создания новых списков слов и добавления слов в набор;
- экран с прогрессом пользователя в изучении и доступом к дополнительным настройкам;
- экран для проверки усвоения слов путем тестирования.

Для добавления наборов слов будет использована «плавающая» кнопка в углу приложения.

Для расширения целевой аудитории приложение будет работать на мобильных устройствах с операционной системой Android 5.0 и выше. По данным статистики от Google версии установленных операционных систем, начиная от 5.0 и выше, на данный момент охватывают более 94% всех используемых мобильных устройств.

Также для использования в приложении функции автоматического перевода текста необходимо, чтобы оно имело подключение к сети Интернет. Приложение будет отслеживать состояние подключения для своевременного уведомления пользователя об его отсутствии и ограничении функционала перевода. Наличие камеры в устройстве не будет являться обязательным условием использования приложения, однако позволит добавлять слова при помощи нее при наличии.

Основными языками программирования для разработки мобильных приложений под Android [8] являются Java и Kotlin. Компания Google назвала Kotlin предпочтительным для платформы Android [22], поэтому он был выбран

для разработки данного приложения. Данное решение было также обусловлено:

- краткостью и лаконичностью языка;
- полезными нововведения (классы данных [3, с. 119-124] и другое);
- возможностью контролировать ссылки на null [3, с. 32-33];
- возможностью продолжать использовать Java-фреймворки и библиотеки, так как они остаются совместимыми с Kotlin.

Android Studio [18] является единственной официально рекомендуемой средой разработки для проектов под систему Android, поэтому разработка велась с помощью данной среды.

В качестве базы данных было решено использовать базу данных SQLite. SQLite – это встроенная библиотека, которая реализует автономный, безсерверный, транзакционный механизм СУБД SQL. Выбор был сделан в пользу данной базы данных, так как в отличие от других она не требует дополнительной установки и сразу может быть использована на любом Android устройстве.

В качестве сервисов для перевода рассматривались популярные сервисы Yandex Cloud Translate [7] и Google Cloud Translation. Оба сервиса предоставляют возможность динамического перевода отдельных слов и целых текстов между большим количеством языковых пар. Наглядно отличия и схожести двух сервисов по переводу отображены в Таблице 1.

Таблица 1. Сравнение Google Cloud Translation и Yandex Cloud Translate

Параметр сравнения	Google Cloud Translation	Yandex Cloud Translate
Количество поддерживаемых языковых Пар	1000+	90+

Продолжение таблицы 1.

1	2	3
Возможность распознавания языка исходного текста	Есть	Есть
Тарифы за перевод и распознавание языка	Бесплатно* 20\$** * Первые 500,000 символов/месяц **За каждые 20 млн символов от 500,000 до 1 миллиарда ***Пробный доступ 3 месяца	Перевод - 447.46 Р* * За каждые 1 млн символов/месяц - 447.46 Р* *За каждые 1 млн символов/месяц ***Пробный доступ 1 месяц

Проанализировав условия и возможности двух сервисов было решено интегрировать в приложение Google Cloud Translation, так как данный сервис предлагает:

- большее количество языковых пар;
- удобную консоль разработчика для отслеживания использования ресурсов сервиса;
- пробный доступ в течении 3 месяцев;
- библиотеку для использования сервиса.

В качестве сервиса для распознавания текста будет использован Google Mobile Vision Text API. Данный сервис поддерживает распознавание языков только с латинским алфавитом. Сервис поддерживается на большинстве устройств Android и не увеличивает размер приложения.

Для реализации представления информации по прогрессу в изучении в виде графиков рассматривались библиотеки MPAndroidChart от разработчика Philipp Jahoda и AnyChart-Android от компании AnyChart [11]. Обе позволяют строить различные виды графиков и кастомизировать их.

Например, при помощи данных библиотеки можно:

- изменить цвет графика;
- изменить количество отображаемых данных за раз;
- применить различные эффекты для отображения одного и того же типа графиков;

- изменить вид подписей значений и другое.

Для разработки приложения была выбрана библиотека MPAndroidChart [19]. Выбор данной библиотеки обусловлен:

- отсутствием требований к наличию лицензии в отличие от AnyChart-Android;

- меньшим весом;

- содержанием всех необходимых методов и вариантов кастомизации, а также достаточным функционалом необходимого для планируемых работ по улучшению приложения.

Выбранные инструменты помогут эффективно реализовать заявленный функционал приложения.

2. Проектирование мобильного приложения для изучения иностранных языков

2.1 Проектирование базы данных

Для хранения добавленных пользователем наборов, слов и прогресса в изучении в приложения используется база данных. Для определения ее ключевых сущностей и обозначения связей была составлена ER-модель [10] (см. Рисунок 9).

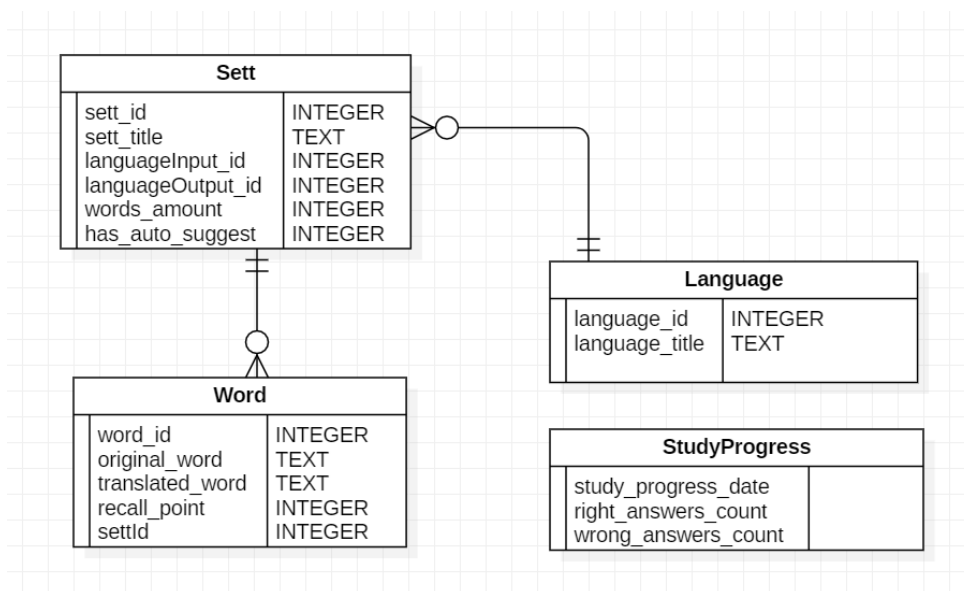


Рисунок 9. ER-модель базы данных

В данной модели присутствуют сущности:

- «Language» – представляет собой сущность языков, используемых в наборах. К атрибутам данной сущности относится: уникальный идентификатор [languageId], название языка [language_title];
- «Set» – представляет собой сущность набора слов. К атрибутам данной сущности относится: уникальный идентификатор [settId], название [settTitle], id языка введенных слов [languageInput_id], id языка для перевода [languageOutput_id], количество слов [wordsAmount], наличие функции автоперевода [hasAutoSuggest];
- «Word» – сущность для хранения слов. К атрибутам данной сущности относится: уникальный идентификатор [id], слово для перевода

[original_word], перевод слова [translated_word], количество баллов, набранных в процессе проверки [recallPoint], внешний ключ на таблицу набора [settId];

– «StudyProgress» – сущность для хранения прогресса в изучении. К атрибутам данной сущности относятся: уникальный идентификатор в виде даты [date], количество правильных ответов [rightAnswers], количество неправильных ответов [wrongAnswers].

В качестве базы данных для разработки данного мобильного приложения используется SQLite. Она поддерживает следующие типы данных:

- TEXT (аналог String в Kotlin);
- INTEGER (аналог Long в Kotlin);
- REAL (аналог Double в Kotlin);
- NUMERIC – для хранения булевых значений, а также времени и дат;
- BLOB - бинарные данные.

Для управления базой данных SQLite на устройствах Android применяется абстрактный класс SQLiteOpenHelper [17]. При использовании данного вспомогательного класса от разработчика скрывается логика, на основе которой принимается решение о создании или обновлении базы данных перед ее открытием. Сам класс SQLiteOpenHelper содержит два обязательных для переопределения метода:

- onCreate() — вызов данного метода происходит при попытке доступа к базе данных, которая еще не была создана;
- onUpgrade() — вызов данного метода происходит при необходимости обновления схемы базы данных. В этом методе можно пересоздать ранее созданную базу данных в onCreate(), установив соответствующие правила преобразования от старой версии к новой.

Также используются другие методы класса:

- `getReadableDatabase()` – для открытия базы данных для чтения;
- `getWritableDatabase()` – для открытия базы данных для записи;

Для реализации этих методов в проекте приложения необходимо создать класс, который будет наследоваться от `SQLiteOpenHelper`. В данном классе будет необходимо реализовать обязательные методы модификации и создания базы данных. В нем же также обычно объявляются различные строковые константы для названия таблиц и полей, однако для удобства они были вынесены в отдельный файл с объектами:

```
//com.example.myapplication.database
object TablesAndColumns {
    object LanguageEntry : BaseColumns {
        const val TABLE_NAME = "language"
        const val COL_LANGUAGE_TITLE = "language_title"
    }
}
```

Это поможет добавить, изменить, удалить уже существующий атрибут базы данных сделав изменения лишь в одном месте.

Для каждой сущности базы данных был использован отдельный репозиторий для управления ею. Все репозитории реализуют интерфейс `IRepository` для того, чтобы иметь реализацию всех основных операций CRUD (добавление, чтение, обновление, удаление):

```
// com.example.myapplication.database.repo
interface IRepository<T> {
    fun create(entity: T): Long
    fun update(entity: T): Long
    fun delete(entity: T): Long
    fun get(id: Long): T?
    fun getAll(): List<T>?
}
```

Пример добавления данных о языке:

```
// com.example.myapplication.database.repo.language
override fun create(entity: Language): Long {
    db = dbHelper.getWritableDatabase
    db.beginTransaction()
    var id = -1L
    try {
        val cv = ContentValues()
        cv.clear();
        cv.put(TablesAndColumns.LanguageEntry.COL_LANGUAGE_TITLE,
```

```

entity.languageTitle)
id = db.insertOrThrow(TablesAndColumns.LanguageEntry.TABLE_NAME,
null, cv)
    db.setTransactionSuccessful()
    } catch (e: Exception) {
Log.d(TAG, "Error while trying to add post to database");
    } finally {
        db.endTransaction()
        return id
    }
}
}

```

Такая реализация функций помогает в последующем не дублировать код операций, а вызывать необходимые из репозитория соответствующей сущности.

Для того, чтобы созданные пользователем данные не исчезли после переустановки, в приложении было настроено ее автоматическое копирование в резервную копию приложений телефона. Для этого необходимо прописать в файле манифеста:

```

android:allowBackup = true
android:fullBackupContent = @xml/my_backup_rules

```

В указанном файле `xml/my_backup_rules` нужно прописать файлы, которые должны быть сохранены в резервной копии приложения, а именно – файл базы данных:

```

<?xml version="1.0" encoding="utf-8"?>
<full-backup-content>
    <include domain="database"
        path="wordsAndProgressDB.db" />
</full-backup-content>

```

Стоит отметить, что функция автоматического резервного копирования и восстановления данных приложения работает только на устройствах с подключенным аккаунтом Google и активации соответствующей функции копирования в настройках телефона [13]. Так как эти данные хранятся на диске Google Drive пользователя, то доступ к ним имеет только владелец аккаунта, что обеспечивает безопасность сохраняемых данных.

2.2 Проектирование пользовательского интерфейса

В Android приложениях создание графического интерфейса пользователя происходит через иерархию объектов View и ViewGroup [1, с. 65-66]. Пример вида иерархии представлен на рисунке 10. Объекты View это виджеты пользовательского интерфейса, такие как кнопки, текстовые поля и другие. В свою очередь ViewGroup это контейнеры, которые не видны пользователю, но они определяют расположение дочерних элементов. Примерами таких контейнеров являются RelativeLayout, LinearLayout, CoordinatorLayout и другие.

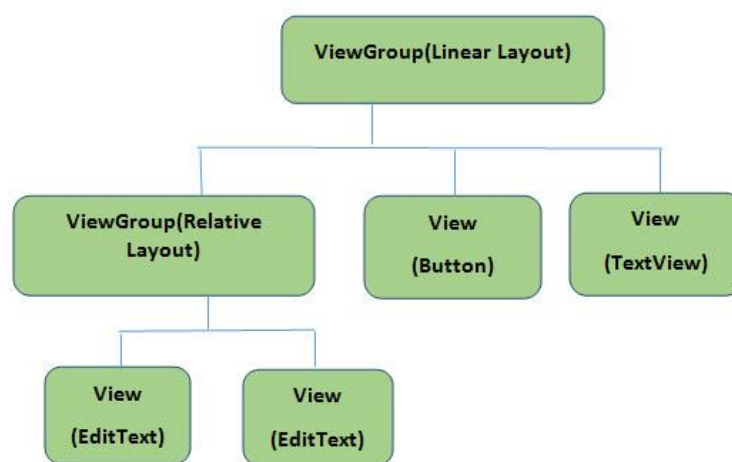


Рисунок 10. Пример иерархии объектов View и ViewGroup

Для создания разметки при определении визуальной составляющей пользовательского интерфейса используются три способа [1, с. 67]:

- объявление элементов интерфейса в XML-файле;
- программное создание элементов управления непосредственно в коде activity;
- сочетанием первых двух способов.

Разметка пользовательского интерфейса в разрабатываемом приложении производится как в XML, так и в коде программы.

Цветовая гамма приложения представлена в фиолетово-розовых тонах и оттенках синего. Розовый цвет использован для выделения различных кнопок и деталей. Фиолетовый как основной цвет фона и панелей навигации.

Фиолетовый цвет был выбран, так как на подсознательном уровне воспринимается как цвет для дорогих, надежных вещей. Сочетание этих цветов позволяет интерфейсу выглядеть стильно и лаконично, при этом грамотно выделяя основные компоненты системы и необходимую информацию, не отвлекая пользователя.

При открытии приложения будет отображаться экран со списком карточек наборов слов (см. Рисунок 11). Сразу под названием набора слов будет располагаться выбранная языковая пара для перевода, а ниже - количество добавленных слов в список. При нажатии на набор будет открываться непосредственно его содержимое (см. Рисунок 12). На этом экране можно будет также редактировать карточки слов и добавлять новые в набор. Также с основного экрана можно будет перейти к созданию нового набора слов при нажатии на плавающую кнопку добавления [14].

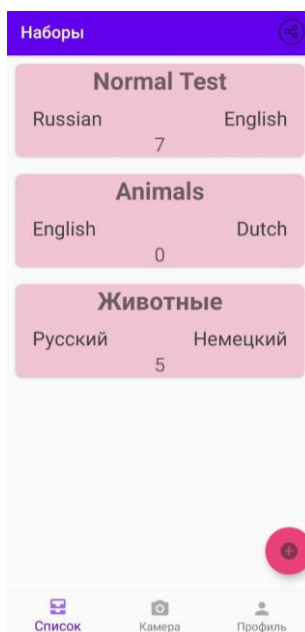


Рисунок 11. Экран со списком карточек наборов слов

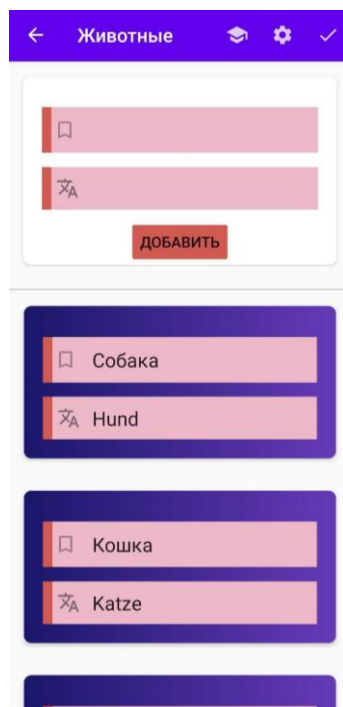


Рисунок 12. Экран со списком карточек набора

При переходе на средней экран будет отображаться картинка с основной камеры устройства для распознавания слов и их последующего добавления в списки (см. Рисунок 13).

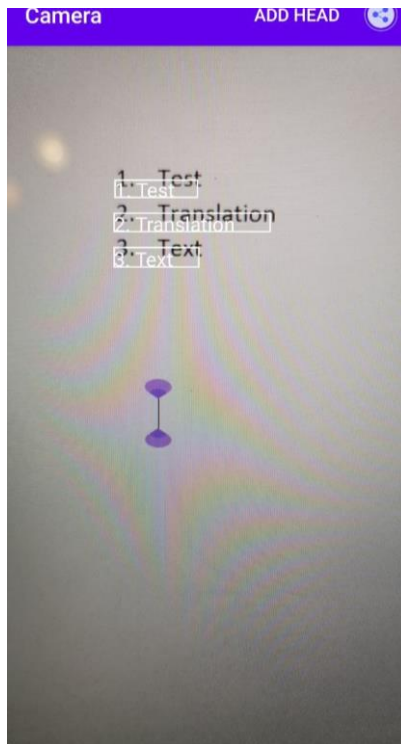


Рисунок 13. Экран с распознаванием

На последнем экране будет отображен профиль пользователя с статистикой прогресса изучения и дополнительные настройки.

Для быстрого перемещения между различными экранами было решено применить `BottomNavigationView` (см. Рисунок 14). `BottomNavigationView` — это нижняя панель навигации. Она позволяет переключаться между экранами приложения в одно касание при помощи закрепленных внизу экрана кнопок. В основном она предназначена для использования в смартфонах, поскольку расположение в нижней части экрана позволяет обеспечить удобный и быстрый доступ для пользователей устройств с небольшими экранами. По рекомендациям панель должна содержать не меньше 3-х, но не больше 5-ти кнопок. Данное условие нам подходит, так как у нас будет использовано 3 главных фрагмента.

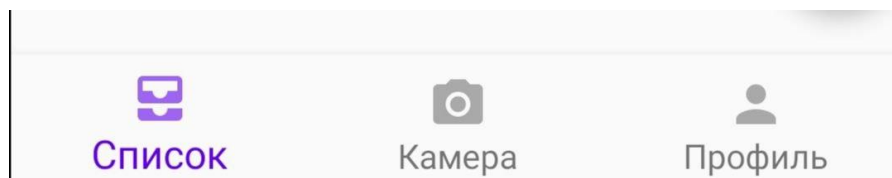


Рисунок 14. Панель нижней навигации

Для реализации трех основных экранов были выбраны фрагменты. Фрагмент представляет собой часть визуального интерфейса приложения, который можно использовать повторно. Фрагмент имеет свой жизненный цикл, однако существует только в контексте `activity` и существовать вне его не может. У фрагмента может быть собственный файл `layout`. Каждая `activity` может иметь несколько фрагментов. Фактически фрагмент — это обычный класс, который наследуется от класса `Fragment`. Как и класс `Activity`, фрагмент может использовать `xml`-файлы `layout` для определения графического интерфейса. Далее показан пример реализации фрагмента с отображением имеющихся списков слов:

```
// layout/fragment_list.xml
<androidx.coordinatorlayout.widget.CoordinatorLayout
    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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ui.list.ListPageFragment">
    <androidx.recyclerview.widget.RecyclerView
```

```

        android:id="@+id/set_list"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </androidx.recyclerview.widget.RecyclerView>
    <com.google.android.material.floatingactionbutton.FloatingAction
Button    android:id="@+id/fab"
        android:contentDescription="@string/add_set_button_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_anchorGravity="bottom|right|end"
        android:src="@drawable/ic_action_name"
        app:layout_anchor="@id/set_list"
        android:layout_marginBottom="20dp"
        app:backgroundTint="#EA417A"/>
</androidx.coordinatorlayout.widget.CoordinatorLayout >

```

В данном коде в CoordinatorLayout мы добавляем необходимые компоненты (см. Рисунок 15):

- RecyclerView для отображения списка наборов слов [21];
- FloatingActionButton – плавающую кнопку для добавления новых списков.

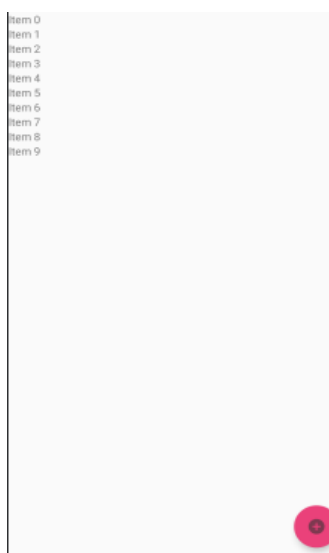


Рисунок 15. Фрагмент для отображением имеющихся наборов слов

Другие фрагменты реализуются по похожей схеме.

Для того, чтобы переключаться между различными фрагментами при помощи нижней панели навигации, необходимо соединить все эти фрагменты в родительском activity – MainActivity. В него добавляем файл с BottomNavigationView и компонент Fragment, где и будут отображаться

нужные нам фрагменты по запросу. Добавим код инициализации **BottomNavigationView** в **MainActivity** для переключения между фрагментами:

```
val navView: BottomNavigationView = findViewById(R.id.nav_view)
// находим файл xml с фрагментом к которому будет прикреплять
BottomNavigationView
val navController = findNavController(R.id.nav_host_fragment)
// Передаем ID каждого меню как set ID
val appBarConfiguration = AppBarConfiguration(
    setOf( R.id.navigation_list, R.id.navigation_camera,
R.id.navigation_profile))
//Настроивает ActionBar возвращенный
[AppCompatActivity.getSupportActionBar] для использования
//с [NavController].
setupActionBarWithNavController(navController,
appBarConfiguration)
navView.setupWithNavController(navController)
```

Для реализации экранов со списком наборов или слов был использован элемент **RecyclerView**. **RecyclerView** предназначен для оптимизации работы со списками. Для его использования, необходимо создать следующее:

- **RecyclerView**, который надо добавить в **layout** нашего экрана (пример добавления показан в коде файла `// layout/fragment_list.xml` выше;
- отдельный **layout** для отображения строк списка;
- адаптер, содержащий данные и связывающий их со списком.

Layout для каждой строки списка представлен компонентом **CardView** внутри которого располагаются нужные элементы (см. Рисунок 16).



Рисунок 16. Пример **CardView**

Адаптер помогает передать данные для отображения в список. В классе **SetAdapter** прописывается код, предназначенный для отображения информации списка набора слов (см. Приложение 1).

Для прохождения тестирования будет открываться новый экран (см. Рисунок 17). На нем будет отображена карточка с текущим вопросом, линией прогресса прохождения по вопросам и вариантами ответа. Линия прогресса прохождения по вопросам будет также окрашиваться в зависимости от данного ответа в зеленый при правильном ответе, в красный – при неверном.

Для отображения верных и неверных ответов, использовались кастомные `TextView`, создаваемые программно:

```
// com.example.myapplication.ui.study
//textView для отображения верных и неверных ответов
for (i in 0 until wordsDisplayed.size) {
    val textView = TextView(this)
    textView.layoutParams = TableLayout.LayoutParams(
        TableRow.LayoutParams.WRAP_CONTENT,
        TableRow.LayoutParams.WRAP_CONTENT, 1f
    )
    textView.id = progressBarIds[i]
    textView.width = 0
    textView.height = (10 * scale + 0.5f).toInt()
    textView.setBackgroundResource(R.drawable.style_points)
    linearLayout.addView(textView)
}
```



Рисунок 17. Экран тестирования

Для ответа на вопрос будет предложено выбрать один из 4 вариантов ответов. После каждого ответа на вопрос карточка с вопросом разворачивается и показывает верный ответ.

3. Разработка мобильного приложения

3.1 Разработка функционала приложения

Перед созданием любого приложения следует определиться с его архитектурой. Самая простая архитектура для Android – приложений – «God Activity» паттерн. Данный паттерн просто подразумевает помещение всего кода в activity, которая относится к одному из экранов. Это делает activity «всемогущим» объектом.

Плюсы данного решения:

- весь код находится в одном месте;
- легкое воплощение.

Минусы данного решения:

- сложность unit-тестирования;
- запутанный код, который зависим от многих частей.

Для решения данных минусов используются другие паттерны как MVP, MVVP и MVC. Основная идея любого из этих паттернов заключается в разделении логики и UI-части приложения. Они помогают убрать лишние зависимости, сделать код более удобным для тестирования.

В данном приложении был применен MVP паттерн [16]. Общие идеи его использования заключаются в использовании:

- Model – уровень данных, обычные классы объектов, которые используются при взаимодействии с View.
- View – уровень отображения, отображает данные, которые получает. View – часть системы, которая видна пользователю, и с которой он взаимодействует.
- Presenter – слой между View и Model. View передаёт ему происходящие события, Presenter занимается их обработкой, при необходимости обращается к Model и возвращает View данные для отображения.

Пример задания интерфейсов View и Presenter для экрана просмотра набора слов:

```
// com.example.myapplication.ui.setCreate
interface SetCreateContract {
    interface Presenter : BasePresenter {
        fun onSaveClicked(
            wordsDisplayed: List<Word>, setTitle: String,
inputLanguage: String,
            outputLanguage: String, hasAutoSuggest: Int)
        fun onLeftSwipe(position: Int)
        fun onTranslate(
            translate: Translate,
            languageTitleAndCode: Map<String, String>,
            originalText: String,
            sourceLanguage: String))
        fun onAddWordClicked(original: String, translated:
String) }
    interface View : BaseView<Presenter> {
        fun updateRecyclerViewInserted(word: Word)
        fun updateRecyclerViewDeleted(position: Int)
        fun showWordInputError()
        fun showUndoDeleteWord(position: Int)
        fun hideKeyboard()
        fun cleanInputFields()
        fun showSuccessSavedToast() } }
```

Паттерн MVP делает:

- activity и фрагменты более легковесными, сводя их работу к отрисовке и обновлению пользовательского интерфейса, обработке событий взаимодействия с пользователем. Благодаря этому приложение становится проще поддерживать;
- написание unit-тестов становится проще.

Для добавления нового набора слов нужно нажать на кнопку добавления внизу экрана с списком наборов. После этого появится диалоговое окно (см. Рисунок 10), в котором будет предложено ввести название набора, языки перевода и включить опцию автоматического перевода.

После создания набора слов перед пользователем откроется экран для добавления слов (см. Рисунок 18). Для сохранения введенных слов нужно

нажать иконку галки. Также нажимая на иконки в верхней строке (см. Рисунок 19), можно перейти к изменению параметров набора или к тестированию.

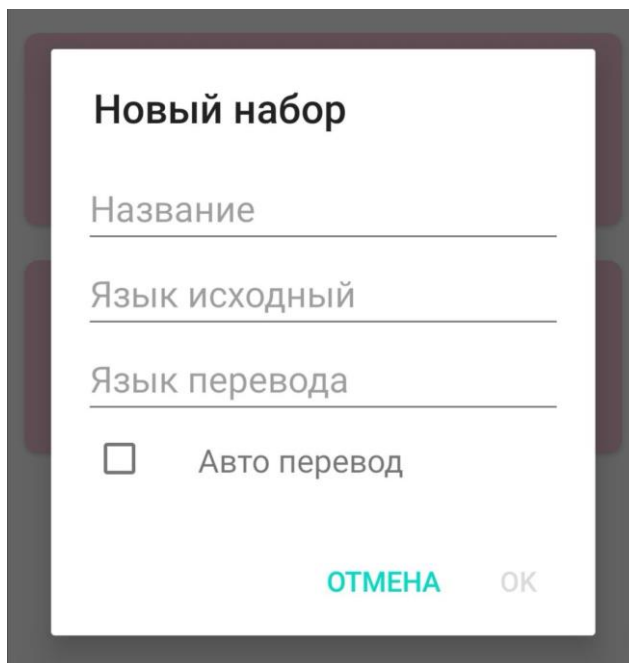


Рисунок 18. Диалог создания набора

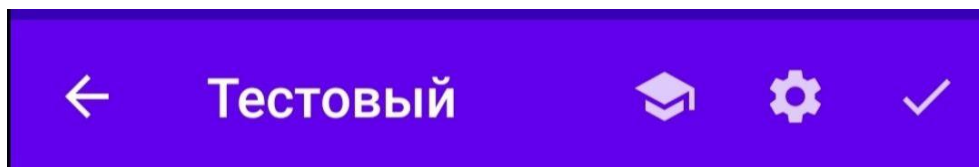


Рисунок 19. Иконки верхней строки

Для организации удобного управления наборами и словами в них в приложение была добавлена поддержка жестов:

```
// com.example.myapplication.ui.setView
override fun onSwiped(viewHolder: RecyclerView.ViewHolder,
direction: Int) {
    val position = viewHolder.adapterPosition
    when (direction) { //свайп влево -> удаление
        ItemTouchHelper.LEFT -> {
            deletedWord = wordsDisplayed[position]!!
            presenter.onLeftSwipe(position)
        }
        //свайп вправо -> копирование
        ItemTouchHelper.RIGHT -> {
            setViewAdapter.notifyItemChanged(viewHolder.adapterPosition)
            val sets = presenter.onRightSwipe()
            showDialog(sets, position)}}}
```

Для того, чтобы удалить набор слов или одно слово необходимо провести по соответствующей карточке влево по экрану. Для отмены действия удаления вызывается Snackbar с кнопкой «ОТМЕНА» (см. Рисунок 20). Если провести по карточке слова вправо по экрану, то появится диалог со списком выбора наборов слов для копирования (см. Рисунок 21).



Рисунок 20. Snackbar для отмены действия

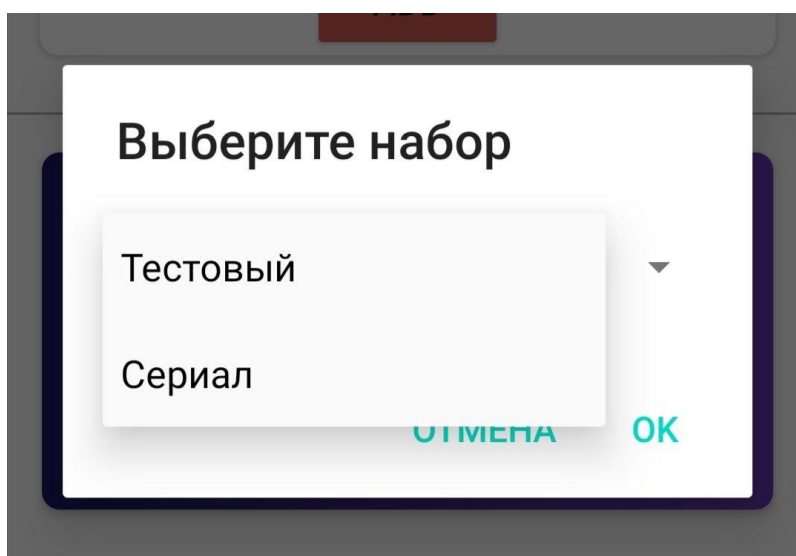


Рисунок 21. Диалог выбора набора слов для копирования

Тестирование запоминания информации было организовано при помощи викторины с 4 вариантами ответов. По этой причине тестирование доступно только при добавлении более 5 карточек в набор. Запуск тестирования производится по иконке внутри набора слов. При попытке запуска тестирования в наборе, где количество карточек меньше 5, вызывается Snackbar с сообщением о минимальном необходимом количестве карточек в наборе (см. Рисунок 22). Правильные ответы во время тестирования отображаются зеленым цветом, неправильные – красным. После каждого ответа карточка переворачивается, показывая верный ответ с обратной стороны. По окончании теста высвечивается количество набранных правильных и неправильных ответов (см. Рисунок 13).

Необходимо минимум 5 карточек для
начала обучения

Рисунок 22. Сообщение при попытке запуска тестирования с недостаточным количеством карточек

При запуске теста после выбора ответа каждая карточка будет поворачиваться обратной стороной, показывая верное значение слова. В конце тестирования будет отображаться сообщением с количеством правильных и неправильных ответов (см. Рисунок 23).

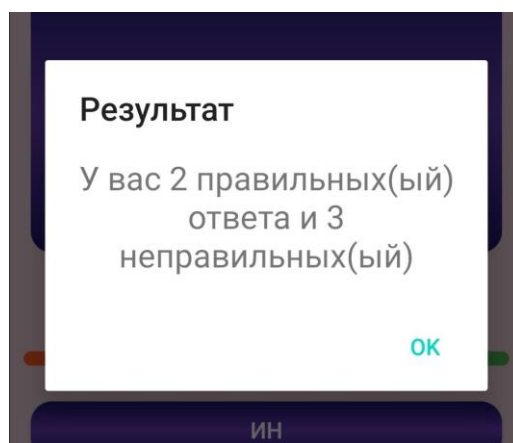


Рисунок 23. Сообщение с результатом

Для отслеживания прогресса изучения на экране «Профиль» при помощи библиотеки MPAAndroidCharts реализовано отображение столбчатого графика (см. Рисунок 23). Данные для графика получаются из соответствующей таблицы «StudyProgress» базы данных в presenter и передаются в фрагмент для отображения. Расчет значений прогресса для каждого дня производится по формуле 3.1.

$$\text{Прогресс} = \frac{\text{правильные ответы}}{(\text{правильные ответы} + \text{неправильные ответов})} * 100\% \quad (3.1)$$

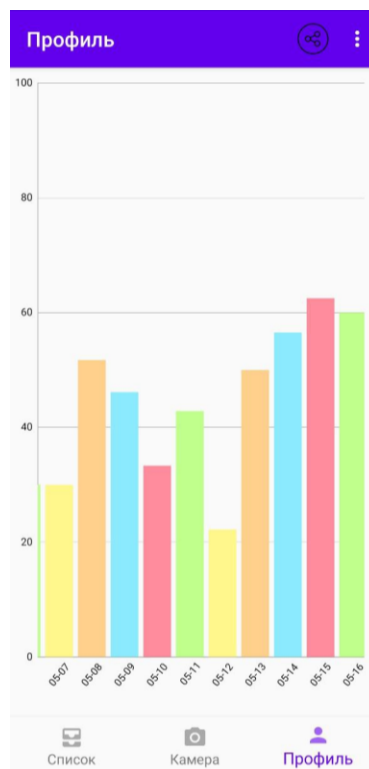


Рисунок 24. График прогресса

Присваивание данных переменной графика:

```
set1 = BarDataSet(values, "Progress")
set1.setColors(*ColorTemplate.VORDIPLOM_COLORS)
set1.setDrawValues(false)
val dataSets: ArrayList<IBarDataSet> = ArrayList()
dataSets.add(set1)
val data = BarData(dataSets)
chart?.data = data
```

Для удобства просмотра графика были также применены функции масштабирования, перемещения по графику при помощи скроллинга, анимирования при просмотре. Доступ к управлению данными функциями осуществляется через меню на вкладке «Профиль» [6, с. 203-214]. Также столбцы графика имеют 5 различных цветов для лучшего визуального разделения столбцов прогресса разных дней. Все это позволит пользователям верно и легко оценивать свой прогресс.

Так как для успешного расширения словарного запаса по системе, основанной на системе Лейтнера необходима регулярность занятий, то была реализована функция настройки напоминания о занятиях [12]. Для настройки напоминаний о занятиях был применен стандартный диалог выбора времени

TimePickerDialog(см. Рисунок 25). После выбора необходимого времени напоминание будет приходить каждый день в выбранное время (см. Рисунок 26). При нажатии на него будет осуществляться переход на экран с списком наборов. Сам показ уведомления происходит через вызов сервиса при помощи класса `AlarmManager`:

```
// com.example.myapplication.ui.profile
val alarmManager =
activity?.getSystemService(AppCompatActivity.ALARM_SERVICE) as
AlarmManager
    alarmManager.setInexactRepeating(
        AlarmManager.RTC_WAKEUP,
        calendar.timeInMillis,
        AlarmManager.INTERVAL_DAY,
        pendingIntent
    )
```

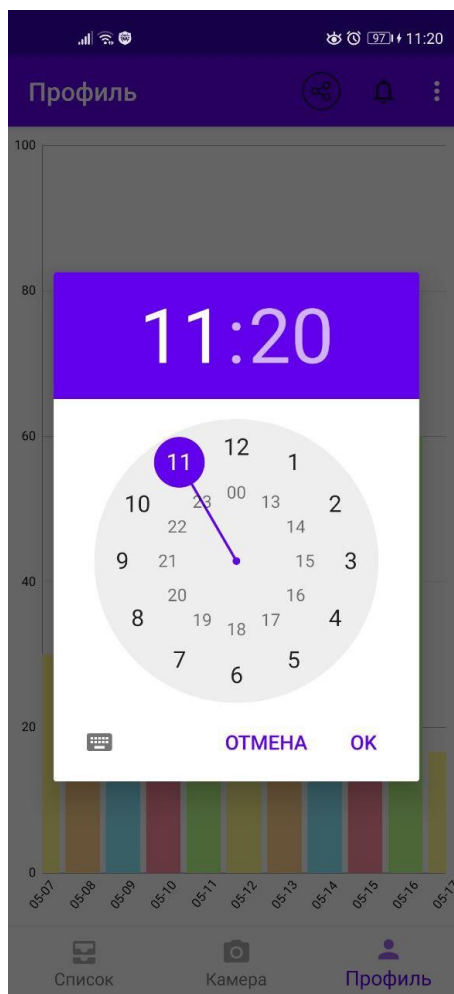


Рисунок 25. Диалог выбора времени напоминания

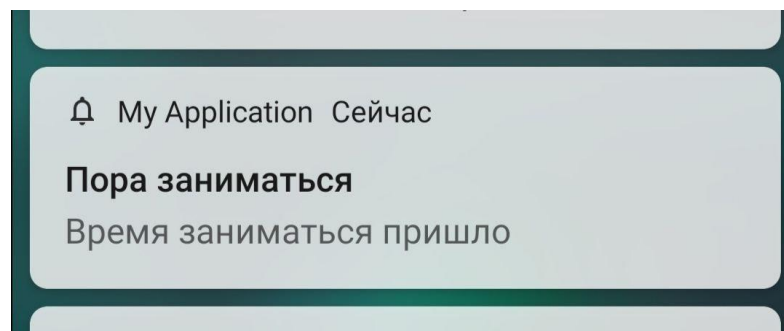


Рисунок 26. Уведомление о занятии

Разработанный функционал приложения позволит грамотно и удобно использовать приложения для достижения максимальных результатов в обучении.

3.2 Внедрение систем перевода и добавления слов

Для максимально продуктивного и комфортного использования приложения в него был интегрирован сервис перевода Google Cloud Translation [15]. Использование сервиса для перевода предложит пользователям большой выбор иностранных языков, которые они могут изучать с помощью данного приложения и повысит привлекательность приложения на фоне конкурентов.

Выбор языков для набора будет осуществляться на этапе создания набора (см. Рисунок 27). Для использования перевода языки нужно вводить на английском языке. При вводе букв будет появляться выпадающий список с предложениями доступных языков для перевода.

Для того, чтобы отобразить список языков, для которых поддерживается перевод необходимо получить их с помощью следующего кода:

```
languages = translate!!.listSupportedLanguages()
```

Для получения языков, на которые будет доступен перевод с выбранного исходного нужно вызвать метод:

```
val languagesTarget =  
translate!!.listSupportedLanguages(Translate.LanguageListOption.  
targetLanguage(  
languageTitleAndCode[editTextInputLang!!.text.toString()] ))
```

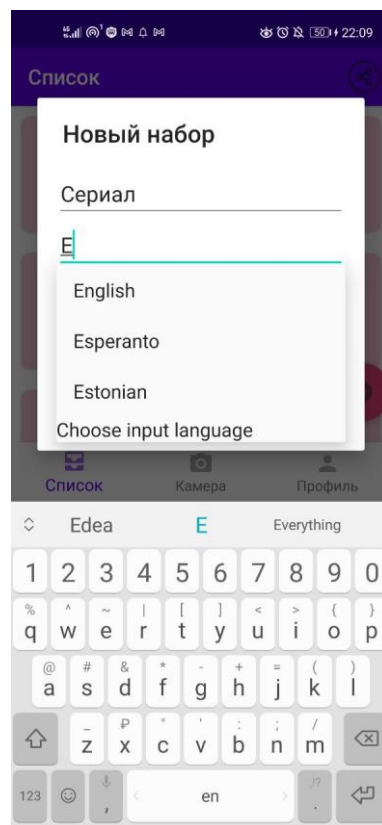



Рисунок 27. Диалог создания набора

Полученные языковые пары будут отображаться при вводе языков при создании и изменении набора слов. Для включения автоматического перевода слов для набора, необходимо выбрать чекбокс «Автоперевод» при создании или изменении набора в диалоговом окне (см. Рисунок 28). Пользователь также сможет создавать наборы слов с переводом на неподдерживаемые сервисом языки, однако он не будет получать автоматические предложения по переводу.

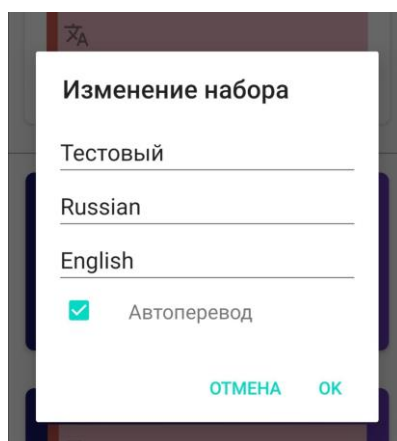


Рисунок 28. Диалог изменения набора

Сам метод для перевода текстов находится в отдельном классе TranslationUtils (см. Приложение 2). Данные о языках перевода передаются в метод из настроек текущего набора слов.

Переведенный текст будет автоматически предложен для ввода при нажатии на текстовом поле предназначенное для введения перевода при условии, что была включена функция автоперевода (см. Рисунок 29).

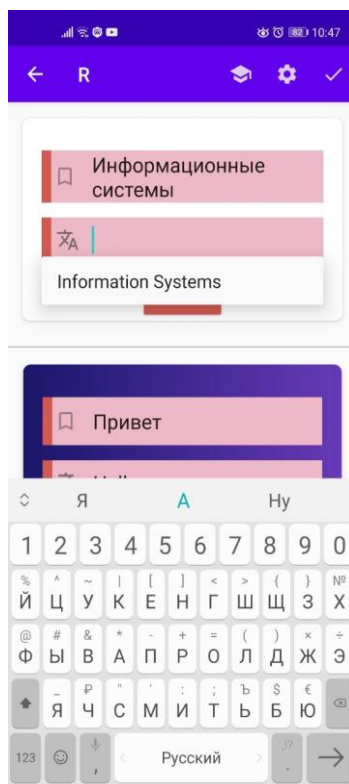


Рисунок 29. Автоперевод слова

Google Cloud Translate предоставляет возможность производить быстрый и качественный перевод для большого количества языковых пар, что существенно повышает привлекательность разрабатываемого приложения для потенциальных клиентов.

Для расширения способов добавления слов помимо стандартного ввода с клавиатуры в приложение было также интегрировано распознавание текста с камеры устройства при помощи Google Mobile Vision Text API. Распознавание поддерживается только для языков с латинским алфавитом. Для того, чтобы пользователь мог идентифицировать часть текста, которую он

хочет добавить в набор слов на экране при распознавании появляются текстовые блоки (см. Рисунок 30).

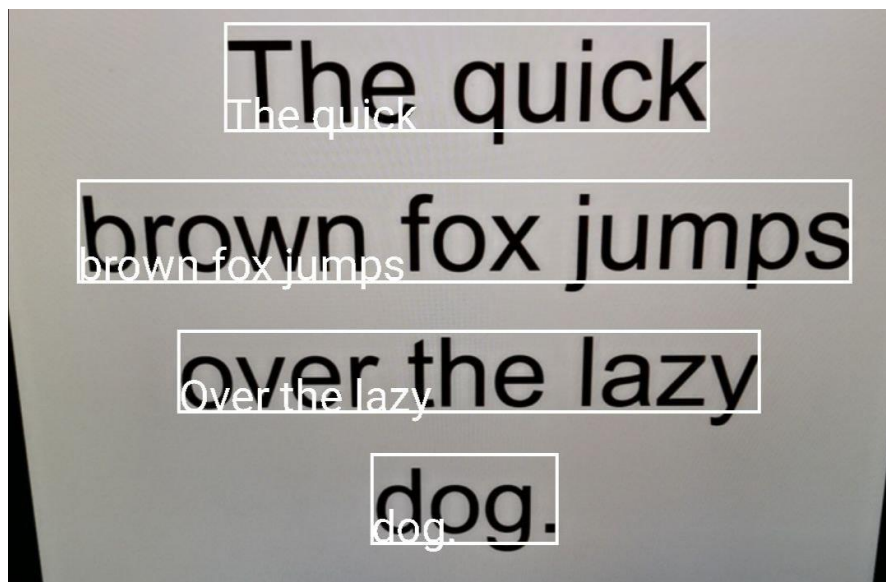


Рисунок 30. Текстовые блоки вокруг распознанного текста.

При нажатии на данные текстовые блоки пользователю будет предложено выбрать набор слов для добавления, и он будет перенаправлен на экран для добавления новых слов (см. Рисунок 31):

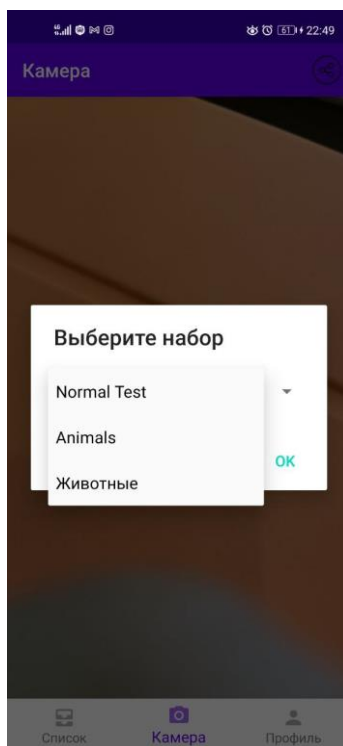


Рисунок 31. Диалог выбора набора для копирования

Также для удобства использования приложения в него было решено добавить плавающую кнопку (см. Рисунок 32), которая позволит при нажатии

добавлять скопированный текст из других приложений напрямую в разработанное приложение. Данная кнопка работает на основе сервиса, который запускается при активации функции по кнопку в верхней панели. Кнопку можно закрепить на любом удобном месте экрана и перемещать ее. Данная функция является новой и не применялась еще ни в одном из приложений- конкурентов.

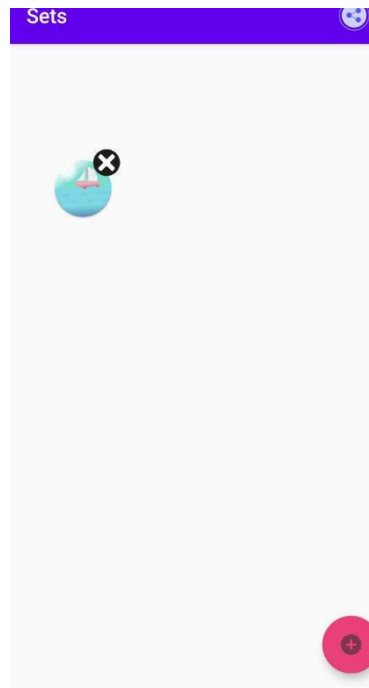


Рисунок 32. Плавающая кнопка для копирования

Функции для получения скопированного текста из буфера обмена:

```
private fun getCopy(): String {
    val copiedText = clipboard.getClipboardText(this)
    return copiedText ?: ""
}

private fun ClipboardManager.getClipboardText(context:
Context): String? {
    if (hasPrimaryClip()) {
        val clip = primaryClip
        if (clip != null && clip.itemCount > 0) {
            val clipboardTextItem = clip.getItemAt(0).coerceToText(context)
            if (clipboardTextItem != null)
                return clipboardTextItem.toString()
        }
    }
    return null
}
```

Использование интегрированного сервиса перевода и различных способов добавления слов значительно повышает привлекательность приложения в глазах пользователей, так как упрощает процесс ввода и перевода слов, предлагая использовать все сервисы в одном месте без необходимости скачивания дополнительных программ.

3.3 Оптимизация работы приложения на различных устройствах

Разрабатываемое мобильное приложение для изучения иностранных языков предназначено для работы на различных Android устройствах. Так как не существует единых стандартов экранов устройств и рекомендуемых к ним разрешений, необходимо убедиться, что отображение интерфейса на экранах с различными диагоналями и разрешениями было правильным.

Тестирование на различных типах устройств проводилось на эмуляторе внутри Android Studio. Данный способ позволяет легко протестировать работу приложения на устройствах без нужды поиска и установки приложения на реальные устройства. Пример отображения интерфейса на 10.1 дюймах показан на рисунке 33. Адаптивность созданного приложения к различным видам экрана достигается при помощи использования параметров «wrap_content» или «match_parent» для элементов и максимального уменьшения использования конкретных величин высоты и ширины.

Приложение предназначено для использования только в портретной ориентации, поэтому в файле манифеста прописан атрибут `android:screenOrientation="portrait"`, который не позволяет перевернуться экрану приложения вне зависимости от системных настроек устройства [2, с.132].

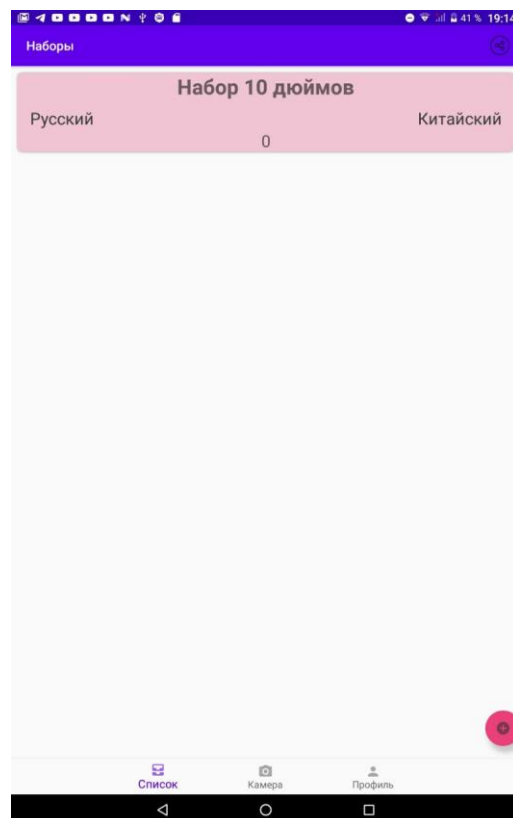


Рисунок 33. Интерфейс на 10.1 дюймах

Приложение предназначено для устройств на базе Android 5.0 и выше. В ходе развития операционной системы некоторые сервисы и используемые методы претерпевали изменения, поэтому в некоторых случаях необходимо указывать версию системы, для которой применяется метод. Например, начиная с Android 8.0 все уведомления должны относиться к определённому каналу, поэтому в коде явно прописывается для какой версии применять следующий код:

```
// com.example.myapplication.ui.profile
private fun createNotificationChannel() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O)
{val name = resources.getString(R.string.channel_name)
val descriptionText =
resources.getString(R.string.channel_description)
val importance = NotificationManager.IMPORTANCE_DEFAULT
val channel = NotificationChannel(CHANNEL_ID, name,
importance).apply {description = descriptionText}
    // Register the channel with the system
val notificationManager: NotificationManager =
        ContextCompat.getSystemService(
            requireContext(),
```

```

        NotificationManager::class.java
    ) as NotificationManager
notificationManager.createNotificationChannel(channel)}

```

Это создаст канал для уведомлений только на устройствах с Android 8.0 и выше.

Также приложение могут использовать люди, для которых родной язык отличен от русского, поэтому была произведена локализация приложения на английский язык. Весь отображаемый текст на элементах приложения переносится в файл строковых ресурсов strings.xml. Данный подход помогает решить проблему дублирования текста вместо его повторного использования, а также упрощает процесс локализации. Упрощение локализации заключается в том, что при локализации на новый язык, создается новый файл с значениями и слова заменяются на новый язык. Приложение при запуске узнает системный язык устройства и применит соответствующий файл к элементам. Пример локализации на русский язык:

```

// values/strings.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="title_list">Список</string>
    <string name="title_camera">Камера</string>
    <string name="title_profile">Профиль</string>
    <string name="set_title_text">Название</string>
...
</resources>

```

По такому же принципу хранятся используемые в приложении цвета в файле colors.xml:

```

// values/colors.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
<color name="colorPrimary">#6200EE</color>
    <color name="colorPrimaryDark">#3700B3</color>
    <color name="colorAccent">#03DAC5</color>
    <color name="green">#4CAF50</color>
    <color name="red">#FF5722</color>
</resources>

```

Такой подход упрощает поддержку и разработку приложения.

После проведения оптимизации приложение заработало корректно на большем количестве устройств.

4. Тестирование приложения

4.1 Тестирование функциональных элементов мобильного приложения

Тестирование приложения является неотъемлемой частью разработки, которое помогает выявить неточности и ошибки в работе приложения после изменения и добавления нового функционала, а также отслеживать работоспособность уже существующих функций и элементов. Применение автоматизированного тестирования помогает обеспечить высокую надежность проверки работы приложения, своевременное выявление возникших ошибок, а также такое тестирование позволяет выполнить большее количество проверок за меньшее время [4, с. 257-258].

Для проведения автоматизированного тестирования необходима имитация действий пользователя приложения. Для этого были выбраны фреймворки Espresso и JUnit. Они позволяют выполнять сложные тесты пользовательского интерфейса как на реальном устройстве, так и на эмуляторе. Для выполнения тестов фреймворкам требуется доступ к исходному коду.

Так как для работы приложения требуется база данных, целесообразно написать тесты, которые позволят отслеживать ее работоспособность после изменений. Для этого были написаны тесты на операции CRUD – создание, чтение, обновление и удаление. Пример проверки на создание:

```
@Test
@Throws(Exception::class)
fun testDbInsertion() {
    val language1 =
com.example.myapplication.entity.Language(0, "Russian1")
    val language2 =
com.example.myapplication.entity.Language(0, "English")
    val langId1 = mLangRepo.create(language1)
    val langId2 = mLangRepo.create(language2)
    val sett = Sett(0, "Test", langId1, langId2, 0, 0)
    val settId = mSetRepo.create(sett)
    val word = Word(0, "Тест2", "Test", 2, settId)
    val wordId = mWordRepo.create(word)
```

```

        assertEquals(1, langId1)
        assertEquals(2, langId2)
        assertEquals(1, wordId)
        assertEquals(1, settId)
    }

```

Основные компоненты Espresso [20]:

1. Espresso – точка входа во взаимодействие с представлениями (через `onView ()` и `onData ()`).
2. ViewAssertions – проверяют состояние объекта и подтверждают, что состояние соответствует критериям;
3. ViewMatchers – находят нужный объект в текущей иерархии представлений;
4. ViewActions – выполняют различные действия с объектами.

Для приложения были составлены различные тесты по обработке нажатия кнопок, введения информации в текстовые поля. Пример тестирования добавления слова в набор:

```

@Test
fun displayAddedWord() {
    val intent = Intent(
        ApplicationProvider.getApplicationContext(),
        SetViewActivity::class.java
    ).putExtra("settId", "1")
    var scenario = activityScenarioRule.scenario
    scenario = launchActivity(intent)
    onView(withId(R.id.original_input)).perform(replaceText("Рассвет"))
    onView(withId(R.id.translated_input)).perform(replaceText("Dawn"), closeSoftKeyboard())
    onView(withId(R.id.recyclerivew_set_create))

    onView(withId(R.id.word_add_button)).perform(click())
    onView(withId(R.id.recyclerivew_set_create))
        .check(matches(atPosition(0,
            hasDescendant(withText("Рассвет")))))
    onView(withId(R.id.recyclerivew_set_create))
        .check(matches(atPosition(0,
            hasDescendant(withText("Dawn")))))
}

```

Запуск тестов производился каждый раз после внесения изменений и добавления новых функций. Такой подход помогал оперативно исправлять появившиеся ошибки в работе программы.

4.2 Тестирование алгоритма распознавания

Для распознавания текста с камеры устройства использовался Google Mobile Vision Text API. Хотя во многом качество распознавания зависит от алгоритма распознавания, разработанного компанией Google, на который повлиять не представляется возможным, качество распознавания также зависит и от настроек, которые будут указаны для камеры устройства перед запуском. Задачей тестирования являлось найти оптимальные параметры настроек камеры и выделения текста для корректной работы на максимальном количестве устройств в различных условиях. Для выяснения сценария использования данной функции был проведен опрос потенциальных пользователей приложения, который выявил, что они планируют использовать распознавание для небольших по размеру блоков текста, поэтому было необходимо выставить высокое разрешение съёмки и включить автофокусировку. Из-за введения карантина во время пандемии опрос пользователей проводился в онлайн-формате. Также для улучшения распознавания был подобран параметр частоты обновления кадров и включена функция автоматического применения вспышки:

```
cameraSource = new CameraSource.Builder(getApplicationContext(),
    textRecognizer)
    .setFacing(CameraSource.CAMERA_FACING_BACK)
    .setRequestedPreviewSize(1280, 1024)
    .setRequestedFps(15.0f)
    .setFlashMode(useFlash ? Camera.Parameters.FLASH_MODE_TORCH :
    null)
    .setFocusMode(autoFocus ?
    Camera.Parameters.FOCUS_MODE_CONTINUOUS_VIDEO : null)
    .build();
```

После применения оптимальных настроек, выявленных в результате тестирования распознавание улучшилось на 15% процентов.

Благодаря примененным настройкам камеры, приложение может обнаруживать текст на отдельных кадрах, используя метод обнаружения в TextRecognizer. Однако, после тестирования приложения при различных сценариях использования было решено добавить чтение текста прямо во время видеосъёмки. Для этого был реализован Processor, который обрабатывает текст, как только он появится на экране. Для этого был реализован интерфейс Detector.Processor и переопределены два метода receiveDetections и release:

```
// com.example.myapplication.ui.cameraPreview;
@Override
public void receiveDetections(Detector.Detections<TextBlock>
detections) {
    graphicOverlay.clear();
    SparseArray<TextBlock> items = detections.getDetectedItems();
    for (int i = 0; i < items.size(); ++i) {
        TextBlock item = items.valueAt(i);
        if (item != null) {
            item.getValue();
            Log.d(TAG, "Текст обнаружен " + item.getValue());
            OcrGraphic graphic = new OcrGraphic(graphicOverlay, item);
        }
    }
}

/**
 * Освобождаем ресурсы связанные с процессом детектирования
 * текста.
 */
@Override
public void release() {
    graphicOverlay.clear();
}
```

Как только распознается текстовый блок, то для каждого создается визуальный блок OcrGraphic (см. Рисунок 34).

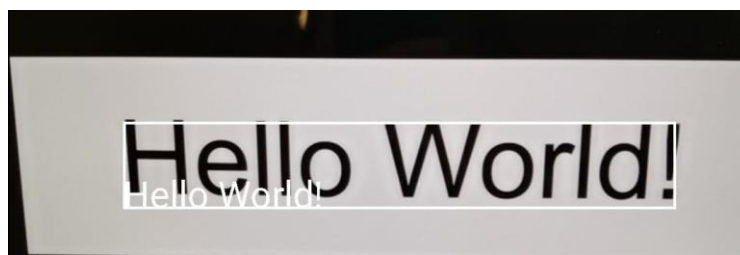


Рисунок 34. Текст с визуальным блоком

Так как чаще всего в приложение будут добавлять отдельные слова или небольшие словосочетания, то отображение текста было решено разбить на несколько строк. Для этого для каждой строки из текстового блока получаем ее местоположение и текст внутри нее:

```
// com.example.myapplication.ui.cameraPreview;
@Override
public void draw(Canvas canvas) {
    if (textBlock == null) {
        return;
    }
    // отрисовка рамки блока текста
    RectF rect = new RectF(textBlock.getBoundingBox());
    rect = translateRect(rect);
    canvas.drawRect(rect, rectPaint);
    // разбиваем текст на несколько строк
    List<? extends Text> textComponents =
textBlock.getComponents();
    for(Text currentText : textComponents) {
        float left =
translateX(currentText.getBoundingBox().left);
        float bottom =
translateY(currentText.getBoundingBox().bottom);
        canvas.drawText(currentText.getValue(), left, bottom,
textPaint);
    }
}
```

После введения данного изменения текст отображается по блокам (см. Рисунок 30).

Для добавления большей функциональности было решено добавить воспроизведение текста при нажатии на текстовый блок. Для реализации был использован TextToSpeech API, встроенный в Android. Для этого в метод касания текстового блока был добавлен вызов функции воспроизведения текста:

```
tts!!.speak(text.value, TextToSpeech.QUEUE_ADD, null, "DEFAULT")
```

Параметры и изменения, внесенные в результате тестирования, повысили качество распознавания и удобство использования приложения.

Заключение

В ходе работы над мобильным приложением были выполнены все поставленные задачи.

Были проанализированы следующие аналогичные приложения «Quizlet», «Flashcards» и «Флэшкарты». Их анализ помог:

- выявить основные востребованные функции в приложениях для изучения иностранных языков;
- ошибки, допущенные в приложениях – конкурентах;
- выявить удачные решения в построении интерфейса системы.

Выявленные достоинства и недостатки других приложений учитывались при разработке приложения. Например, было решено использовать «плавающую» кнопку в углу приложения для создания наборов карточек, так как это помогало освободить место на экране и предоставить быстрый доступ к часто используемой функции.

Была определена целевая аудитория приложения и, исходя из этого, составлены требования к функционалу приложения:

- правильно реализованный подход к обучению при помощи системы, основанной на методе Лейтнера;
- удобная система управления, хранения и напоминания;
- различные способы добавления слов в приложение;
- отслеживание прогресса обучения.

Были определены инструменты для создания мобильного приложения. В качестве них были выбраны:

- Kotlin – язык программирования;
- Android Studio – среда разработки;
- SQLite – база данных;
- Google Cloud Translation – сервис перевода;
- Google Mobile Vision Text – сервис распознавания текста;

- Espresso и JUnit [9] – библиотеки для тестирования.
- MPAndroidChart – для построения графика прогресса обучения.

С помощью данного набора инструментов можно полностью реализовать весь необходимый функционал приложения и протестировать его.

При проектировании системы хранения внимание было уделено корректности составления таблиц и связей между ними, наличию необходимых полей для содержания требуемой информации. Итоговая схема базы данных полностью соответствует требованиям по реализуемому функционалу приложения.

При проектировании интерфейса принимались во внимание интерфейсы рассмотренных приложений – конкурентов. Были применены правила сочетания цветов и расставления акцентов с учетом комфорта использования приложения. Также принимались во внимание результаты анализа других приложений. Спроектированный интерфейс предоставляет удобное и быстрое перемещения между 4 главными экранами приложения и адаптируется к различным размерам экранов устройств.

В ходе разработки функционала приложения была спроектирована удобная система копирования и удаления карточек со словами при помощи жестов. Также в приложение было интегрировано несколько способов добавления текста:

- стандартный ввод при помощи клавиатуры;
- распознавание текста с камеры устройства;
- прямое копирование из других приложений при помощи плавающей кнопки.

Для максимально эффективного использования приложения в него был интегрирован сервис перевода Google Cloud Translation, который позволяет моментально переводить текст на сотни языковых пар.

Тестирование пользователя, по изученным словам, в приложении производится на основе системы Лейтнера при помощи выбора правильного

ответа из 4 предложенных и отображения ответа на вопрос при помощи поворота карточки. Прогресс в изучении отображается по дням в виде столбчатого графика во вкладке профиля. Также для выработки системы повторения новых и старых слов в приложении была реализована настраиваемая система напоминаний.

Тестирование приложения проводилось как во время, так и после разработки приложения. Тестирование во время разработки помогало выявлять возникшие ошибки на раннем этапе, а после – помогало довести приложение к необходимому уровню качества работы на различных устройствах.

Тестирование алгоритма распознавания помогло выявить оптимальные параметры необходимые для максимальной корректности распознавания текста с камеры устройства. Оптимальные настройки улучшили качество распознавания на 15% по сравнению с начальными. Также во время тестирования было принято решение о распознавании текста не на отдельных статических кадрах, а на видео потоке, что позволило сделать приложение еще более удобным.

Разработанное приложение соединяет в себе:

- удобный интерфейс с минималистичным дизайном интерфейса, который не отвлекает пользователя от основной цели использования, грамотно выделяет необходимые элементы и корректно отображается на различных размерах экранов;
- удобную систему управления и навигации;
- интегрированный сервис перевода для быстрого перевода слов на сотни языковых пар внутри приложения;
- сервис по распознаванию текста с камеры устройства для быстрого добавления слов;
- «плавающая» поверх других приложений кнопка, которая помогает в один клик добавить скопированный текст из других приложений в

нужный набор слов. Данная функция является новой и не имеет аналогов ни в одном из приложений конкурентов.

Созданное приложение планируется использовать как для личных нужд, так и выложить в публичный доступ в агрегатор приложений Google Play. Приложение планируется поддерживать и выпускать к нему обновления с новыми функциями, которые помогут стать приложению еще более удобным и полезным. Так как приложение использует платные сервисы перевода, то доступ к нему будет предоставляться по подписке 150 руб./мес.

Список использованных источников

1. Голощапов А.Л. Google Android: программирование для мобильных устройств [Текст]. - 2 изд. - СПб.: БХВ-Петербург, 2012. - 448 с.: ил
2. Дейтел, П. Android для разработчиков [Текст]/ П. Дейтел, Х. Дейтел, Э. Дейтел. — СПб.: Питер, 2015. — 384 с.: ил. — «Библиотека программиста».
3. Жемеров Д., Kotlin в действии [Текст] / Исакова С. пер. с англ. Киселев А. Н.- М.: ДМК Пресс, 2018. - 402 с.: ил.
4. Куликов С. Тестирование программного обеспечения [Текст]/ С. Куликов. — Москва: ЕРАМ Systems, 2017. — 298 с.
5. Лейтнер, С. Метод интервальных повторений [Текст] = So lernt man lernen/ С. Лейтнер ; пер. с немец. А. Г. Торицина. — Москва: Перо, 2019. — 108 с.
6. Хашими С. Разработка приложений для Android [Текст]: практическое руководство/ Хашими С., Коматинени С., Маклин Д. - СПб.: Питер, 2011. - 736 с.:ил.
7. Как начать работать с Translate [Электронный ресурс]. – Режим доступа: <https://cloud.yandex.ru/docs/translate/quickstart>. Дата обращения: 11.11.2020.
8. Общие сведения о платформе Android [Электронный ресурс]. – Режим доступа: <https://developer.android.com/guide>. Дата обращения: 25.10.2020.
9. Основы JUnit [Электронный ресурс]. – Режим доступа: <https://divancoder.ru/2017/06/junit-basic/>. Дата обращения: 06.04.2021.
10. Что такое ER-диаграмма [Электронный ресурс]. – Режим доступа: <https://www.lucidchart.com/pages/ru/erd-диаграмма>. Дата обращения: 20.11.2020.

11. AnyChart for Android [Электронный ресурс]. – Режим доступа: <https://github.com/AnyChart/AnyChart-Android>. Дата обращения: 02.11.2020.
12. Create a Notification [Электронный ресурс]. – Режим доступа: <https://developer.android.com/training/notify-user/build-notification>. Дата обращения: 12.03.2021.
13. Data backup overview [Электронный ресурс]. – Режим доступа: <https://developer.android.com/guide/topics/data/backup#:~:text=of%20the%20ring tone,-,Backup%20options,the%20user's%20Google%20Drive%20account>. Дата обращения: 23.03.2021.
14. Floating Action Buttons [Электронный ресурс]. – Режим доступа: <https://guides.codepath.com/android/floating-action-buttons>. Дата обращения: 10.01.2021.
15. How to use Google Translate API in Android Studio projects? [Электронный ресурс]. – Режим доступа: <https://medium.com/@yeksancansu/how-to-use-google-translate-api-in-android-studio-projects-7f09cae320c7>. Дата обращения: 25.01.2021.
16. Getting Started with MVP (Model View Presenter) on Android [Электронный ресурс]. – Режим доступа: <https://www.raywenderlich.com/7026-getting-started-with-mvp-model-view-presenter-on-android#toc-anchor-007>. Дата обращения: 26.02.2021.
17. Local Databases with SQLiteOpenHelper [Электронный ресурс]. – Режим доступа: <https://guides.codepath.com/android/local-databases-with-sqliteOpenHelper>. Дата обращения: 01.10.2021.
18. Meet Android Studio [Электронный ресурс]. – Режим доступа: <https://developer.android.com/studio/intro>. Дата обращения: 10.11.2020.
19. MPAndroidChart [Электронный ресурс]. – Режим доступа: <https://github.com/PhilJay/MPAndroidChart>. Дата обращения: 5.11.2020.
20. Espresso basics [Электронный ресурс]. – Режим доступа: <https://developer.android.com/training/testing/espresso/basics>. Дата обращения: 20.03.2021.

21. RecyclerView [Электронный ресурс]. – Режим доступа: <https://metanit.com/java/android/5.11.php>. Дата обращения: 23.01.2021.

22. Using Kotlin for Android Development [Электронный ресурс]. – Режим доступа: <https://kotlinlang.org/docs/reference/android-overview.html>. Дата обращения: 01.10.2020.

Код класса SetAdapter для отображения наборов слов через RecyclerView

```
// класс адаптер для отображения наборов слов через RecyclerView
class SetAdapter internal constructor(
    context: Context?, onSetListener: OnSetListener,
    private val sets: LinkedHashMap<Sett, List<String>>
) :
    RecyclerView.Adapter<SetAdapter.ViewHolder>() {
    private val inflater: LayoutInflater =
        LayoutInflater.from(context)
    var onSetListener: OnSetListener = onSetListener
    //передаем шаблон строки
    override fun onCreateViewHolder(parent: ViewGroup, viewType:
        Int): ViewHolder {
        val view: View =
            inflater.inflate(R.layout.sets_list_item_veiw, parent, false)
        return ViewHolder(view, onSetListener)
    }
    // передаем информацию для отображения
    override fun onBindViewHolder(holder: ViewHolder, position:
        Int) {
        val listValues: List<List<String>> =
            ArrayList<List<String>>(sets.values)
        val listKeys: List<Sett> = ArrayList<Sett>(sets.keys)
        val langList: List<*> = listValues[position]
        holder.setTitle.text = listKeys[position].settTitle
        holder.languageInput.text = langList[0].toString()
        holder.languageOutput.text = langList[1].toString()
        holder.wordsAmount.text =
            listKeys[position].wordsAmount.toString()
    }

    // возврат размера
    override fun getItemCount(): Int {
        return sets.size
    }

    class ViewHolder(view: View, onNoteListener: OnSetListener) :
        RecyclerView.ViewHolder(view), View.OnClickListener {
        val setTitle: TextView =
            view.findViewById(R.id.set_title)
        val languageInput: TextView =
            view.findViewById(R.id.language_input)
        val languageOutput: TextView =
            view.findViewById(R.id.language_output)
        val wordsAmount: TextView =
```

```

view.findViewById(R.id.words_amount)
    var mOnSetListener: OnSetListener = onNoteListener
    override fun onClick(view: View) {
        mOnSetListener.onSetClicked(adapterPosition)
    }
    init {
        itemView.setOnClickListener(this)
    }
}
interface OnSetListener {
    fun onSetClicked(position: Int)
}
}

```

Код класса TranslationUtils для перевода слов

```

class TranslationUtils {
    companion object {
/**
 * Метод для перевода текста
 * @param translate переменная библиотеки класса перевода
 * @param languageTitleAndCode Map поддерживаемых языков
перевода и их кодов
 * @param originalText текст для перевода
 * @param sourceLanguage язык с которого необходим перевод
 * @param targetLanguage язык на который нужно перевести
 * @return переведенный текст
 */
        fun translate(
            translate: Translate,
            languageTitleAndCode: Map<String, String>,
            originalText: String,
            sourceLanguage: String,
            targetLanguage: String
        ): String {
            if (originalText != "") {
                if (sourceLanguage != "" &&
                    languageTitleAndCode.containsKey(sourceLanguage)) {
                    return if (targetLanguage != "" &&
                        languageTitleAndCode.containsKey(targetLanguage)) {
                        try {
                            val translation: com.google.cloud.translate.Translation =
                                translate.translate(originalText,
                                    Translate.TranslateOption.sourceLanguage(languageTitleAndCode[sourceLanguage]),
                                    Translate.TranslateOption.targetLanguage(languageTitleAndCode[targetLanguage]))
                            translation.translatedText;
                        } catch (e: Exception) {
                            ""
                        }
                    } else {
                        try {
                            val translation: com.google.cloud.translate.Translation =
                                translate.translate(originalText,
                                    Translate.TranslateOption.sourceLanguage(languageTitleAndCode[sourceLanguage]),
                                    Translate.TranslateOption.targetLanguage(languageTitleAndCode["English"]))
                        }
                        translation.translatedText;
                    } catch (e: Exception) {
                        ""
                    }
                }
            }
        }
    }
}

```

