Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Хабаровский государственный университет экономики и права»

Факультет управления и технологий

Кафедра информационных систем и технологий

«ДОПУСТИТЬ К ЗАЩИТЕ»

Зав. кафедрой _____
(наименование)
_____

_____/_____/
подпись Ф.И.О.

«____» _____ 20 ___г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по направлению 09.03.03 «Прикладная информатика»,
профиль «Корпоративные информационные системы»
по теме «Разработка мобильного приложения для изучения английского языка студентами ФГБОУ ВО «ХГУЭП»

Студент группы ПИ(б)-81 _____ _____ Н.В. Фетисов
номер группы дата подпись

Научный руководитель _____ _____ _____ _____ С. И. Белозерова
уч. степень, уч. звание дата подпись

Нормоконтролёр _____ _____ _____ _____ С. И. Белозерова
уч. степень, уч. звание дата подпись

Хабаровск 2022

УТВЕРЖДАЮ:

Зав. кафедрой

Информационные системы и технологии

(наименование)

_____ /Чуйко О.И./

подпись Ф.И.О.

«____» _____20___ г.

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ХАБАРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ЭКОНОМИКИ И ПРАВА»

Задание по подготовке ВКР студ. факультета управления и технологий, ПИ(б)-81,

(факультет, группа, Ф.И.О.)

Фетисова Никиты Владимировича

Тема работы «Разработка мобильного приложения для изучения английского языка студентами ФГБОУ ВО «ХГУЭП»
утверждена приказом по ХГУЭП № 1880-студ от «16» декабря 2021 г.

Источники и исходные данные к работе отчет по производственной (преддипломной) практике

В теоретическом разделе до «____»_____20____г. Определить и проанализировать проблему, представить ее решение.

В аналитическом разделе до «____»_____20____г. выявить существующие аналоги, описать объект автоматизации, описать технологии создания мобильного приложения.

В рекомендательном разделе до «____»_____20____г. сформулировать основные проблемы и методы улучшения качества процесса обучения иностранному языку для студентов ФГБОУ ВО «ХГУЭП», разработать мобильное приложение, выполнить расчет экономической эффективности.

Перечень графического материала: таблиц 3 рисунков 41 приложений 5

Сроки сдачи студентом законченной работы «____»_____20____г.

Дата выдачи задания «_____» _____ 20____ г.

Научный руководитель работы _____ /_____/

подпись Ф.И.О.

Задание принято к исполнению «____» _____ 20___ г.

Студент _____ /_____/

подпись Ф.И.О.

РЕФЕРАТ

Выпускная квалификационная работа «Разработка мобильного приложения для изучения английского языка студентами ФГБОУ ВО «ХГУЭП» 86 с., 4 разд., 41 рис., 3 табл., 40 источн., 5 прил.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, REACT NATIVE, REALM, FIGMA, БАЗА ДАННЫХ, JAVASCRIPT, FLOW, КРОССПЛАТФОРМЕННОСТЬ.

Объект исследования – проблемы изучения иностранного языка и способы их решения.

Предмет исследования – процесс разработки мобильного приложения для изучения английского языка студентами ФГБОУ ВО «ХГУЭП».

Цель работы – проектирование и разработка мобильного приложения для изучения английского языка студентами ФГБОУ ВО «ХГУЭП».

В процессе исследования была выполнена постановка проблемы и представлен способ ее решения. Определен технический инструментарий для разработки кроссплатформенного мобильного приложения. Выполнены следующие задачи: создание дизайна пользовательского интерфейса, проектирование базы данных, разработка мобильного приложения. Произведен расчёт экономической эффективности.

Результат исследования – готовое к эксплуатации мобильное приложение для изучения английского языка».

Область применения – изучение английского языка студентами ФГБОУ ВО «ХГУЭП».

Экономическая эффективность заключается в том, что внедрение результатов исследования упростит процесс изучения английского языка студентами университета, посредством значительной экономии ими времени и усилий, что в свою очередь скажется на качестве получения высшего образования.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Технологическая сингулярность. Этот термин лаконично описывает квинтэссенцию современного прогресса. Прогресса в сферах развития новых технологий, образования, получения информации.

Три эти сферы ставят сложные задачи для любой альма-матер и потому ключевая идея данной выпускной квалификационной работы – предоставление программного обеспечения с целью упростить выполнение этих задач, а конкретнее, «Разработка мобильного приложения для изучения английского языка студентами Хабаровского государственного университета экономики и права».

Актуальность темы обусловлена причинами ее выбора. Которые представляют собой следующее:

1. Важность изучения английского языка.

2. Сложность его изучения студентами.

3. Желание повысить качество и понизить сложность в освоении вышеуказанной дисциплины.

В прошлом, настоящем и обозримом будущем английский язык де-факто Лингва франка для всего мирового сообщества и потому важность его изучения не ставится под сомнение.

И для того, чтобы достигнуть цели данной выпускной квалификационной работы необходимо выполнить следующие задачи:

1. Исследовать предметную область, обозначить проблему и представить решение.

2. Определить концептуальные особенности проекта.

3. Сформулировать перечень требований к системной архитектуре и функционалу приложения.

4. Разработать дизайн мобильного приложения.

5. Определить технические средства для разработки мобильного приложения, а также базы данных.

6. Спроектировать базу данных.

7. Разработать мобильное приложение.

8. Выполнить расчет экономической эффективности.

Первая глава данной выпускной квалификационной работы представляет собой постановку, анализ и решение проблемы.

Во второй главе описаны существующие аналоги мобильного приложения, обоснована цель его проектирования и приведены требования к ПО. Выбран и обоснован инструментарий для разработки, а также спроектирована база данных.

В третьей главе описан пользовательский интерфейс мобильного приложения, а также структурные и технические особенности. Помимо этого, рассмотрено взаимодействие с серверами, предоставляющими API открытых словарей.

В четвертой главе произведен расчет экономической эффективности приложения.

В приложениях представлен исходный код разработанного мобильного приложения.

Во время выполнения выпускной квалификационной работы была использована техническая литература, а также ресурсы сети интернет и документация используемого инструментария для разработки приложения.

1 Теоретическое обоснование проблемы

1.1 Постановка проблемы

Проектирование и разработка мобильного приложения для изучения английского языка осуществляется для обучающихся «Хабаровского государственного университета экономики и права» и проводилась во время прохождения преддипломной практики на кафедре ИСТ этого учебного заведения [17].

Студенты ФГБОУ ВО «ХГУЭП» получают образование не только по дисциплинам кафедры, на которой обучаются, но и по учебным программам других кафедр и факультетов.

И в процессе обучения, учащиеся могут сталкиваться со сложностями в освоении учебных программ университета. Эти сложности, как правило, носят самый разный характер, от тривиальных до специфических.

Во время исследования предметной области для написания данной выпускной квалификационной работы, которое проводилось в процессе прохождения практики, было выявлено, что одна из дисциплин университета лидирует по уровню сложности ее освоения среди студентов и это английский язык, преподаваемый кафедрой иностранных языков и межкультурной коммуникации, которая декларирует следующие цели изучения дисциплины [2]:

— повышение исходного уровня владения иностранным языком, достигнутого на предыдущей ступени образования;

— овладение студентами необходимым и достаточным уровнем знаний для решения социально-коммуникативных задач в различных областях бытовой, культурной, профессиональной и научной деятельности при общении с зарубежными партнерами;

— повышение уровня учебной автономии, способности к самообразованию.

А также следующие задачи дисциплины:

— формирование и развитие умений устных форм общения в официальных и неофициальных ситуациях общения с носителями языка, необходимых для выступлений с сообщениями и докладами, а также осуществления личных профессиональных контактов;

— совершенствование письменных речевых умений в конкретных ситуациях делового общения;

— овладение стилистическими особенностями речевого поведения в рамках социокультурной и профессионально-деловой сфер общения.

Ввиду всех вышеперечисленных целей и задач, студентам университета не всегда в силу различных обстоятельств удается освоить программу кафедры на достаточно высоком уровне.

1.2 Анализ и решение проблемы

Во время изучения не только английского, но и любого иностранного языка обучающиеся сталкиваются, как правило, с идентичным спектром проблем [20]:

— неспособность думать на английском языке;

— трудность преодоления языкового барьера;

— трата большого количества времени и при этом получение низкого результата, что отрицательно сказывается на желании изучать язык;

— сложность восприятия английской речи на слух и трудности в произношении;

— сложность запоминания английских слов, де-факто ставшая одной из самых значительных проблем в изучении языка.

В разрешении вышеперечисленных проблем способен помочь вспомогательный инструментарий. Это могут быть техники запоминания слов, подходы, помогающие избавиться от сложности восприятия речи и многие другие способы.

Потому целью, данной выпускной квалификационной работы стало предоставление такого инструментария для студентов ФГБОУ ВО «ХГУЭП», созданное в рамках научной направленности кафедры ИСТ – проектирование и разработка информационных систем.

Данная ИС призвана облегчить процесс изучения английского языка, а также сэкономить время студентов всех направленностей, оптимизировав процесс получения необходимой информации.

Так как проектируемая система прежде всего ориентирована на студентов, то она должна быть доступной в виде программного обеспечения для такого устройства, которое в любой момент есть под рукой у каждого – мобильный телефон.

И потому разрабатываемое ПО представляет из себя мобильное приложение, акцент функционала которого прежде всего направлен на избавление от ранее выявленного спектра проблем. Тем самым приложение будет решать следующие задачи:

1. Сокращение расходуемого пользователями времени на обучение.
2. Тренировка произношения английских слов.
3. Повышение качества запоминания слов.

Последний пункт признан особо важным, ведь как показывает практика основным камнем преткновения в изучении иностранного языка является процесс запоминания новых слов.

Эту проблему стоит рассмотреть подробнее на примере «Кривой забывания Эббингауза», представленной на рисунке 1.

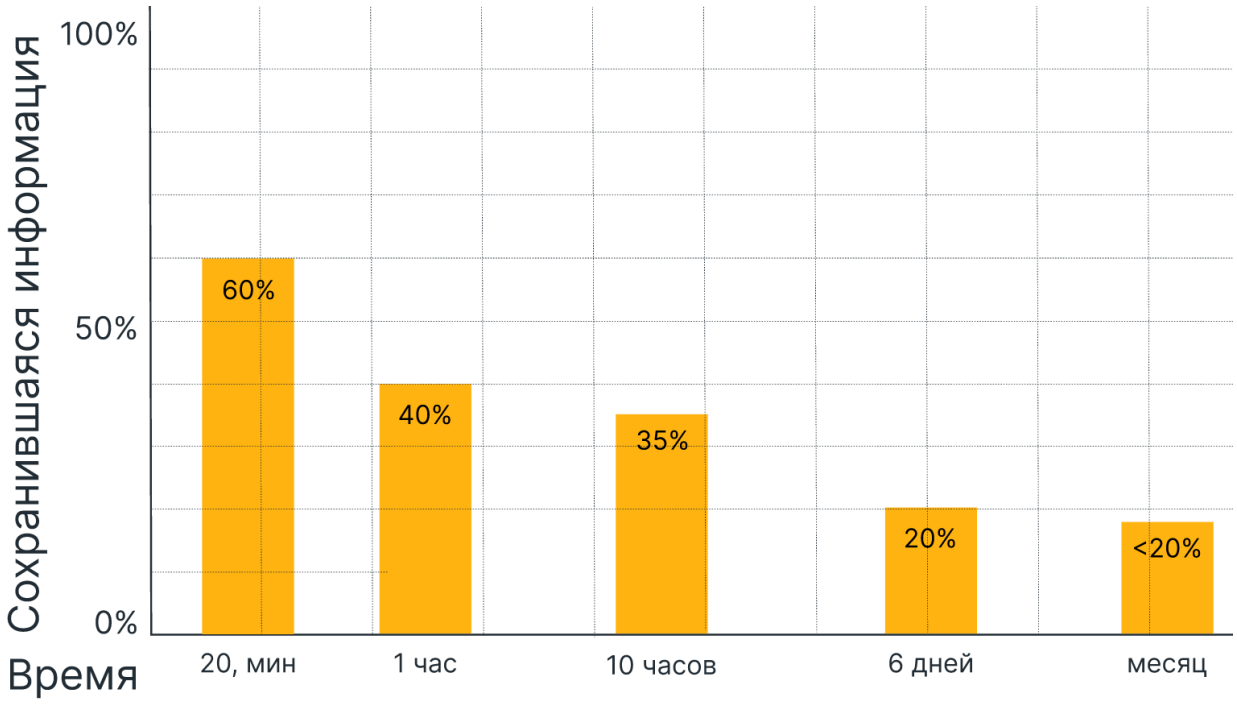

Рисунок 1 – Кривая забывания

Кривая Эббингауза демонстрирует то, насколько быстро человек забывает изученный материал [15].

В ходе опытов было установлено, что после первого безошибочного повторения слов забывание идёт вначале очень быстро. Уже в течение первого часа забывается до 60 % всей полученной информации, через 10 часов после заучивания в памяти остаётся 35 % от изученного.

Далее процесс забывания идёт медленно и через 6 дней в памяти остаётся около 20 % от первоначально изученной информации, столько же остаётся в памяти и через месяц.

Для закрепления информации в памяти навсегда предлагается использовать следующий рецепт повторений:

— через 10 минут;

— затем – через 1 час;

— через 5 часов;

— 1 день;

— 5 дней;

— 25 дней.

Благодаря этой методологии можно добиться высоких результатов в пополнении словарного запаса и потому одна из опций проектируемого приложения будет заключаться в напоминании пользователю повторять изученный ранее материал с указанными выше интервалами времени.

Также в процессе прохождения преддипломной практики были проанализированы такие дисциплины как:

— информационные технологии;

— экономическая теория;

— маркетинг;

— деловые коммуникации;

— история государства и права;

— социология права;

— инвестиционный менеджмент;

— конституционное право и многие другие.

Все эти дисциплины, преподаваемые в университете обладают собственным тезаурусом терминов, знание которых полезно не только на родном языке, но и на иностранных.

Этот анализ позволил спроектировать одну из ключевых особенностей разрабатываемого приложения – предоставление пользователям специализированных словарей, каждый из которых содержит список слов английского языка полезных как в изучении, так и в применении каждой из дисциплин, преподаваемых в университете.

2 Проектирование программного обеспечения

2.1 Анализ и сравнение с текущими решениями

Ввиду общемировой распространённости английского языка рынок мобильных приложений, а также рынки программного обеспечения для других устройств, предлагают множество решений, призванных помочь пользователям в изучении этого языка.

Разброс программного обеспечения в этой отрасли идет от самых простых приложений для запоминания слов до полноценных школ английского языка, предоставляющих целые экосистемы для полного погружения в среду цифрового билингвизма.

Приложением, обладающим схожими концептуальными и техническими решениями с приложением, описываемым в данной выпускной квалификационной работе, является «Reword Английский».

Reword – это мобильное приложение для изучения слов английского языка, распространяемое на платформах IOS и Android, и представленное на рисунке 2 [19].

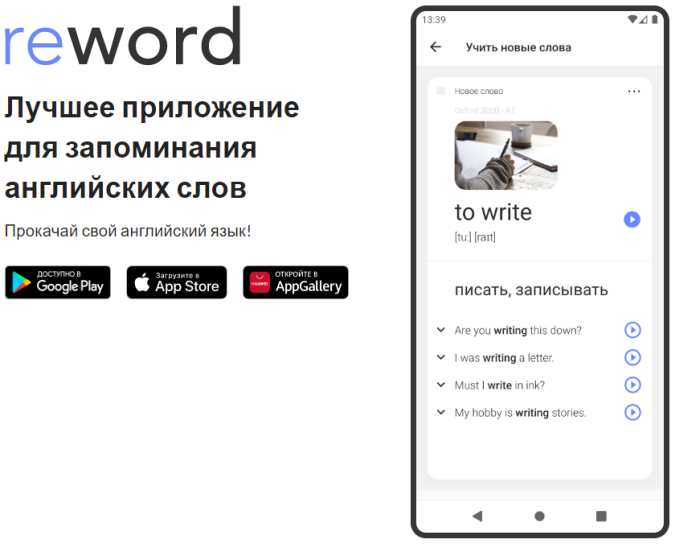

Рисунок 2 – Официальный сайт Reword

Reword позволяет пользователям добавлять свои слова с помощью автоматического дополнения слов из разных источников как это представлено на рисунках 3 и 4.

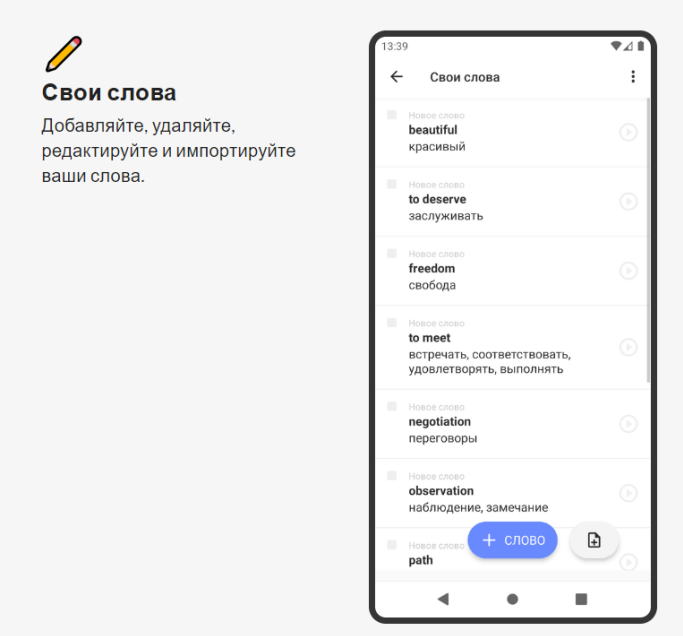

Рисунок 3 – Добавление слов в reword

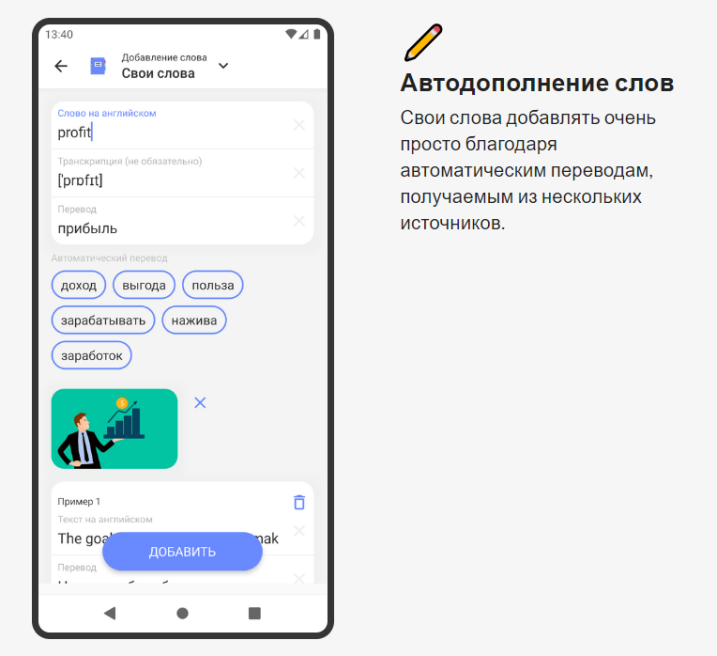

Рисунок 4 – Автоматическое дополнение слов

Однако, данный аналог не лишен недостатков и ключевым из них является функционал добавления слов. Это подразумевает следующие ограничения:

— невозможность добавления новых слов на любом языке кроме английского;

— отсутствие воспроизведения произношения слова на экране добавления;

— ограничения на количество изучаемых слов и наличие таргетированной рекламы в приложении;

— небольшое количество автоматически предоставляемых переводов;

— отсутствие группировки переводов слова по частям речи.

Последний из вышеперечисленных пунктов является довольно весомым минусом, ввиду того что слово может иметь довольно большой набор значений и потому группировка по частям речи позволяет не только категоризировать возможные переводы, но и лучше понимать как-то или иное слово может использоваться в речи.

И потому одно из требований к приложению, разрабатываемому в рамках данной выпускной квалификационной работы является ликвидация всех описанных недостатков.

2.2 Выбор основных технологий реализации ПО

Перед началом проектирования необходимо определить какая платформа является целевой для данного ПО, так как от этого зависит то, насколько приложение будет доступно среднестатистическому пользователю, а также в случае выхода приложения на рынок за пределами предприятия, то насколько высока будет конвертируемость пользователей в платящих клиентов.

Статистика по загрузке и доходам приложений для двух доминирующих целевых платформ на мобильном рынке IOS и Android за 2020 год представлена на рисунке 5 [23].

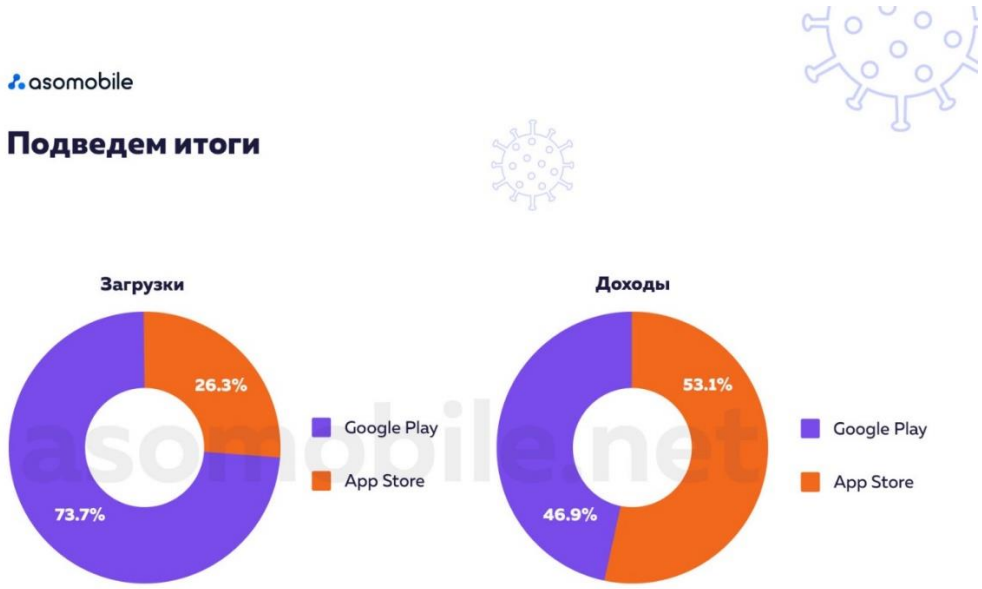

Рисунок 5 – Статистика загрузки приложений по двум целевым платформам

Как видно на рисунке 5 73.7% загрузок мобильных приложений приходится на Google play, то есть на Android. Из этого следует вывод о том, что эта платформа обладает большей популярностью среди пользователей.

Но в то же самое время доходы с приложений загружаемых с App Store, то есть IOS, превалируют доходы приложений на Android в отношении 53.1% к 46.9%. Из этого можно заключить, что конвертируемость пользователей в клиентов готовых покупать услуги, предоставляемые продуктом у пользователей на IOS выше, чем на Android.

К тому же ввиду того, что приложение, создаваемое в рамках данной ВКР предназначено для студентов университета, среди которых примерно в равной степени популярными являются обе платформы, было принято решение от том, что приложение должно быть доступно как на IOS, так и на Android.

Данное требование негативно отражается на сложности разработки приложения ввиду того, что:

— для разработки под каждую операционную систему необходима своя команда инженеров, так как каждая из платформ обладает собственной спецификой и технологией разработки;

— в процессах проектирования, создания и поддержания продукта на всех стадиях его жизненного цикла необходимо одновременно добавлять новый функционал отдельно для каждой из платформ, что увеличивает время разработки и вероятность появления артефактов проектирования и конструирования на стадии пост релиза приложения, а это может отрицательно сказаться на желании клиентов пользоваться продуктом;

— разработанное приложение должно быть размещено в магазинах для каждой из платформ. А так как у каждого магазина существуют свои требования и правила к размещению, то это затрудняет одновременное обновление.

Для того чтобы эффективно управлять сложностью разработки ПО необходимо разрешить вышеописанные трудности. Этого можно добиться следующим способом. Вместо того, чтобы разрабатывать приложение на каждую из платформ отдельно используя нативный инструментарий, то есть Java/Kotlin под Android и Swift/Objective-C под IOS, можно прибегнуть к кроссплатформенным технологиям, позволяющим использовать одну и ту же кодовую базу сразу для нескольких платформ.

Благодаря такому подходу удастся не только сэкономить время на разработке приложения, но и уделить больше внимания деталям, связанным с концептуальными особенностями приложения, вместо акцента на специфических моментах разработки под каждую платформу.

На данным момент существуют две популярные и широко поддерживаемые в сообществе программистов технологии:

1. Flutter.

2. React Native.

Flutter — комплект средств разработки и фреймворк с открытым исходным кодом для создания мобильных приложений под Android и iOS, веб-

приложений, а также настольных приложений под Windows, macOS и Linux с использованием языка программирования Dart, разработанный и развиваемый корпорацией Google [30].

React Native — это кроссплатформенный фреймворк с открытым исходным кодом для разработки нативных мобильных и настольных приложений на JavaScript и TypeScript [33].

React Native поддерживает такие платформы как Android, Android TV, iOS, macOS, Apple tvOS, Web, Windows и UWP, позволяя разработчикам использовать возможности библиотеки React вне браузера для создания нативных приложений, имеющих полный доступ к системным API платформ.

Оба фреймворка обладают плюсами и минусами, нивелирующими друг друга, но в ходе анализа было принято решение о разработке приложения на React Native, ввиду обладания опытом использования фреймворка React, на основе которого и построен React Native. Это позволит сократить время на разработку и избежать ошибок.

React Native предполагает использование языка программирования JavaScript, который обладает динамической слабой системой типов.

С одной стороны, такая типизация удобна, поскольку делает процесс разработки программного обеспечения более простым, так как разработчику не нужно задумываться о типах сущностей в его коде, нет необходимости прибегать к использованию статического полиморфизма в связи с тем, что язык осуществляет проверку типов во время исполнения кода.

Однако это влечет за собой некоторые неудобства в особенностях отлаживания кода и поиска ошибок, что в свою очередь может привести к появлению undefined behavior (неопределенному поведению) и артефактов в продуктовой версии приложения.

Для того чтобы этого избежать существуют три инструмента:

1. Инструменты для строгой типизации в JS, а также подобный инструментарий предоставляемый React Native.

2. TypeScript – язык программирования, созданный корпорацией Microsoft и позиционируемый как средство разработки приложений, расширяющее возможности JavaScript [38].

3. Flow – надстройка над JavaScript, позволяющая использовать в нем статическую сильную типизацию.

Так как Flow разработан той же командой, что и React Native, было принято решение использовать именно его в текущем проекте. Это поможет сделать процесс разработки мобильного приложения более качественным, удобным и позволит отлавливать ошибки прямо во время написания кода, что благоприятно отразится на качестве создаваемого ПО.

2.3 Выбор базы данных для приложения

Как было заявлено ранее, основная концепция разрабатываемого мобильного приложения основана на запоминании новых слов английского языка.

Следовательно, приложение должно предоставлять функционал для создания персонализированных словарей и содержащиеся в них данные требуется сохранять для повторного использования и потому необходима база данных.

И тут выбор неочевиден, так как встает вопрос, какой тип системы управления базой данных подходит лучше всего для реализации функционала мобильного приложения.

И в пространстве вариантов существуют два возможных решения, реляционная система и нереляционная. Так как база данных в приложении будет использоваться преимущественно для хранения загружаемых пользователем слов, то схема базы данных будет сравнительно небольшой и для такой задачи отлично подойдет нереляционная база данных.

СУБД такого класса не используют язык запросов SQL, и, следовательно, не нуждаются в ORM для связывания базы данных с парадигмой объектно-ориентированного программирования.

Процесс работы с реляционными базами данных безусловно можно реализовать и без использования ORM, но в таком случае это не так удобно и займет больше времени.

Одной из самых популярных СУБД с большим сообществом разработчиков является Realm от MongoDB.

Realm – объектно-ориентированная система управления базами данных с открытым исходным кодом, созданная преимущественно для разработки ПО для операционных систем IOS и Android [37].

Ее прямой конкурент реляционная СУБД SQLite написанная на языке C и впервые представленная в 2000 году.

Однако, Realm обладает следующими преимуществами в сравнении с SQLite:

— простота написания исходного кода;

— легко справляется со сложными операциями;

— предоставляет большой API;

— данные представлены в качественной документации.

2.4 Требования к проектированию

Разработка программного обеспечения – процесс, включающий в себя множество этапов. За последние 25 лет ученые выявили следующие составляющие разработки ПО [3]:

— определение проблемы;

— выработка требований;

— создание плана конструирования;

— разработка архитектуры ПО, или высокоуровневое проектирование;

— детальное проектирование;

— кодирование и отладка;

— блочное тестирование;

— интеграционное тестирование;

— интеграция;

— тестирование системы;

— корректирующее сопровождение.

Проблема была определена в четвертом пункте первой главы, данной выпускной квалификационной работы.

Требования же к самому проектируемому мобильному приложению будут рассмотрены в следующем пункте данной главы.

Текущим пунктом в разработке ПО является конструирование.

Конструирование – процесс кодирования или программирования, а также детального проектирования, создания плана, отладки, тестирования и интеграции системы.

Конструирование подразумевает под собой следующие задачи:

1. Проверка выполнения условий, необходимых для успешного конструирования.

2. Определение способов последующего тестирования кода.

3. Проектирование и написание классов и методов, в том случае, если речь идет об ООП.

4. Создание и присвоение имен переменным и именованным константам.

5. Выбор управляющих структур и организация блоков команд.

6. Блочное тестирование, интеграционное тестирование и отладка собственного кода.

7. «Шлифовка» кода путем его тщательного форматирования и комментирования.

8. Интеграция программных компонентов, созданных по отдельности.

9. Оптимизация кода, направленная на повышение его быстродействия, и снижение степени использования ресурсов.

Во время проектирования и разработки приложения все вышеописанные пункты должны быть приняты к сведению и соблюдены.

Не менее важной частью требований к разработке ПО является детальное и качественное понимание разработчиком технологий, используемых при проектировании системы.

Потому перед началом разработки ПО на React Native, важно понимать его внутреннюю структуру.

Приложение на React Native состоит из двух частей [35]:

1. Код написанный на JavaScript.

2. Нативный код целевой платформы.

Код React Native приложения представляет собой отдельные элементы, объединенные в компоненты, которые затем отрисовываются в пользовательский интерфейс.

Компоненты включают в себя:

— логику;

— состояние;

— разметку.

Логика – это базовый функционал, описанный внутри компонента и в зависимости от того какого вида, компонент функциональный или классовый, может быть представлена функциями или методами.

Состояние – это набор некоторых данных, хранящихся внутри компонента и имеющих свойство изменяться со временем.

Разметка – это то, чем описывается пользовательский интерфейс. Она представляет из себя JSX – JavaScript Syntax Extension (Расширение синтаксиса JavaScript) и представлена на рисунке 6 [11].

```
render(): Node {
    return (
        <SafeAreaProvider>
            <ThemeContext.Provider
                value={{
                    theme: this.state.theme,
                    toggleTheme: this.state.toggleTheme
                }}
            >
                <Navigation />
            </ThemeContext.Provider>
        </SafeAreaProvider>
    );
}
```

Рисунок 6 – JSX разметка

В своей сущности она представляет из себя абстракцию над стандартной XML разметкой интерфейса операционных систем мобильных устройств.

После компиляции каждое JSX выражение представляет из себя классический вызов функции, результатом которого является объект JavaScript.

Нативный код целевой платформы в контексте React Native приложения – это код написанный на Java, Swift или Objective-C, который контактирует с API операционной системы и позволяет приложению функционировать.

JavaScript код React Native обладает возможностью взаимодействовать с Native API посредством технологии Bridge, проксирующей данные между этими двумя частями приложения.

Это работает следующим образом. В тот момент, когда происходит запуск приложения, операционная система создает основной поток и предоставляет его приложению. Основной поток создает поток исполнения JavaScript кода и теневой поток, который также известен как теневое дерево.

Задача теневого потока вычислять элементы, описанные в JS и перенаправлять их в нативную часть приложения. И как итог приложение

функционирует согласно следующему алгоритму – JS создает представления, затем они вычисляются теневым деревом, перенаправляются в основной поток приложения и отрисовываются в пользовательском интерфейсе.

Также необходимо понимать устройство нереляционной базы данных, используемой в приложении.

Структура Realm имеет следующий вид, представленный на рисунке 7.

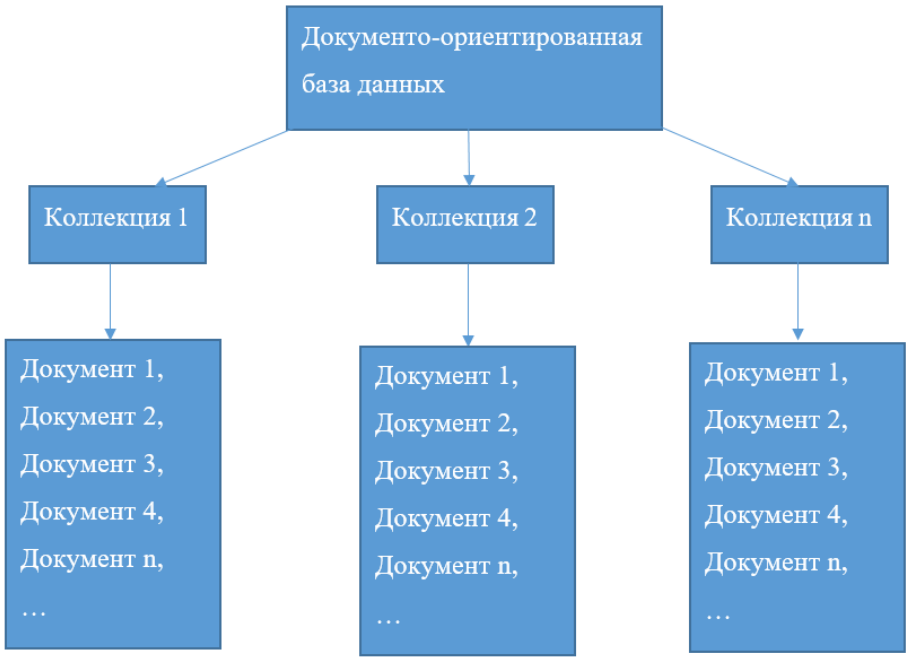

Рисунок 7 – Структура документоориентированной базы данных

Документоориентированная СУБД – СУБД, специально предназначенная для хранения иерархических структур данных (документов) и обычно реализуемая с помощью подхода NoSQL (нереляционный). В основе документно-ориентированных СУБД лежат документные хранилища, имеющие структуру дерева [14].

База данных такого типа состоит из коллекций, а каждая коллекция из документов. Документ можно представить, в виде объекта, хранящего некоторую информацию. В некотором смысле он подобен строкам в реляционных СУБД, где хранится информация об отдельном элементе.

Помимо всего вышеперечисленного важно понимать, что при проектировании ПО, исправление ошибки в требованиях, обнаруженной на этапе проектирования архитектуры, обычно обходится втрое дороже, чем исправление аналогичной ошибки, найденной на этапе выработки требований. Такая же ошибка, обнаруженная при кодировании, обходится дороже уже в 5–10, при тестировании системы – в 10, а после выпуска программы – в 10-100 раз [3].

2.5 Требования к разрабатываемому приложению

Перед проектированием мобильного приложения необходимо сформировать перечень требований, которому оно должно соответствовать.

Требования к инструментарию разработки:

— программное обеспечение должно быть реализовано с помощью фреймворка React Native;

— база данных должна быть создана посредством СУБД MongoDB Realm;

— кодовая база JS должна быть написана с использованием технологии Flow для применения к ней статической сильной типизации.

Требования к функционалу приложения:

Приложение должно:

— предоставлять пользователю главный экран, включающий в себя все основные разделы;

— обладать навигацией по всем основным экранам;

— предусматривать персонализацию пользовательского интерфейса;

— предоставлять пользователю возможность создания персонализированных словарей;

— предоставлять пользователю специализированные словари, которые должны содержать термины дисциплин, изучаемых в университете;

— предоставлять возможность добавления своих слов в словари;

— предоставлять пользователю автоматический перевод слов;

— при автоматическом переводе слов должно группировать варианты перевода по частям речи;

— обладать функцией воспроизведения аудиозаписей произношения слов как при автоматическом переводе, так и на экране списка слов в словаре;

— предоставлять возможность создания не только англо-русских, но и русско-английских словарей.

Требования к пользовательскому интерфейсу:

Приложение должно включать в себя:

— главный экран;

— экран поиска слов;

— экран с перечнем словарей;

— экран со списком слов для каждого словаря;

— экран добавления новых слов;

— экран настроек приложения.

2.6 Подготовка среды разработки

Детальное проектирование и конструирование мобильного приложения осуществляется в редакторе кода Visual Studio Code.

Visual Studio Code – редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений [39].

За тестирование работы приложения отвечает Emulator Android Studio, который эмулирует работу операционной системы Android и тем самым служит средой инсталляции и запуска приложения. Также понадобиться IDE Android Studio, как для инсталляции эмулятора, так и для удобного редактирования нативного кода для Android.

Android Studio — интегрированная среда разработки (IDE) для работы с платформой Android [25].

Для разработки приложения недостаточно одной только стандартной библиотеки React Native, нужен также дополнительный комплект зависимостей, который будет отвечать за создание необходимого функционала приложения. В React Native все зависимости устанавливаются в файл package.json, который располагается в корне каталога исходного кода приложения и выглядит как это представлено на рисунках 8-9.

```
"scripts": {
    "android": "react-native run-android",
    "android:release": "react-native run-android --variant=release",
    "ios": "react-native run-ios",
    "start": "react-native start",
    "test": "jest",
    "lint": "eslint ."
},
"dependencies": {
    "@react-native-async-storage/async-storage": "^1.17.6",
    "@react-native-picker/picker": "^2.4.1",
    "@react-navigation/bottom-tabs": "^6.3.1",
    "@react-navigation/native": "^6.0.10",
    "@react-navigation/native-stack": "^6.6.2",
    "@realm/react": "^0.3.0",
    "axios": "^0.27.2",
    "parse-json": "^6.0.2",
    "react": "17.0.2",
    "react-native": "0.68.1",
    "react-native-get-random-values": "^1.8.0",
    "react-native-modal": "^13.0.1",
    "react-native-navigation-bar-color": "^2.0.1",
    "react-native-picker-select": "^8.0.4",
    "react-native-safe-area-context": "^4.2.5",
    "react-native-screens": "^3.13.1",
    "react-native-sound": "^0.11.2",
    "react-native-splash-screen": "^3.3.0",
    "react-native-svg": "^12.3.0",
    "react-native-swipe-list-view": "^3.2.9",
    "react-native-vector-icons": "^9.1.0",
    "realm": "^10.20.0-beta.5"
},
```

Рисунок 8 – Скрипты и основные зависимости

```
"devDependencies": {
    "@babel/core": "^7.12.9",
    "@babel/eslint-parser": "^7.17.0",
    "@babel/preset-flow": "^7.16.7",
    "@babel/runtime": "^7.12.5",
    "@react-native-community/eslint-config": "^2.0.0",
    "babel-jest": "^26.6.3",
    "babel-plugin-module-resolver": "^4.1.0",
    "eslint": "^7.32.0",
    "eslint-plugin-flowtype": "^8.0.3",
    "flow-bin": "^0.176.2",
    "jest": "^26.6.3",
    "metro-react-native-babel-preset": "^0.67.0",
    "react-test-renderer": "17.0.2"
},
```

Рисунок 9 – Зависимости периода разработки

В верхней части файла, изображенной на рисунке 8 находится json объект «scripts», в нем располагаются свойства хранящие скрипты для запуска из командной строки.

Ниже находится объект «dependencies», он содержит все основные зависимости приложения, используемые как во время разработки, так и во время работы приложения.

На рисунке 9 изображен объект «devDependencies», содержащий зависимости, исключительно периода разработки и не нужные после запуска приложения.

Для инсталляции зависимостей используется пакетный менеджер Yarn.

Yarn – это пакетный менеджер созданный для среды Node.js, как альтернатива npm [40].

После определения требований и установки необходимых библиотек необходимо спроектировать структуру базы данных. Так как основная задача БД – хранить слова, добавленные пользователем, то ее схема не будет отличаться сложностью.

Как был описано в предыдущем пункте, СУБД Realm состоит из документов и в контексте разрабатываемого приложения они выглядят как это представлено на рисунках 10–11.

```
static schema: WordSchema = {
    name: 'Word',
    primaryKey: '_id',
    properties: {
        _id: 'objectId',
        word: 'string',
        translate: 'string',
        transcription: 'string',
        pronunciation: 'mixed',
        partsOfSpeech: '{}',
        dictionaries: 'Dict[]',
        createdAt: 'date',
        containBy: {
```

Рисунок 10 – Схема документа слова

На рисунке 10 представлена схема документа слова. Она отвечает за то каким образом, и какая информация о слове будет храниться в локальной базе данных приложения.

Схема слова имеет следующие свойства:

— id, уникальный идентификатор слова;

— word, текст слова;

— translate, перевод слова;

— transcription, транскрипция слова;

— pronunciation, произношение слова;

— dictionaries, свойство указывает на то, что слово может содержаться сразу в нескольких словарях, но при этом занимать всего одну запись в памяти устройства;

— partsOfSpeech, части речи, к которым принадлежит слово;

— createdAt, дата создания.

Схема документа словаря представлена на рисунке 11.

```
static schema: DictSchema = {
    name: 'Dict',
    primaryKey: '_id',
    properties: {
        _id: 'objectId',
        key: 'string',
        name: 'string',
        wordsAmount: 'int',
        icon: 'string',
        words: 'Word[]',
        createdAt: 'date',
```

Рисунок 11 – Схема документа словаря

Схема документа словаря, устроена схожим образом со схемой документа слова, каждый словарь может содержать множество слов, но при этом каждое слово может быть включено во множество словарей.

Для инициализации базы данных существует ES6 модуль, функционирующий по паттерну singleton (одиночка), благодаря этому объект базы данных инстанцируется только один раз, при первом импорте модуля, а при последующих импортах на экземпляр базы данных просто передается ссылка.

На рисунке 12 изображено создание экземпляра класса Realm, с передачей ему свойства schema, содержащим массив схем, которые будут использоваться в качестве шаблонов при создании коллекций документов.

```
import Realm from 'realm';
import { Word, Dict } from './schemas';

const realm: Realm = new Realm({
    path: 'anglicanumdb',
    schema: [Dict.schema, Word.schema]
});

export default realm;

export { Word, Dict };
```

Рисунок 12 – Инстанцирование класса Realm

Последним пунктом подготовки среды разработки является проектирование того, каким образом будет реализован функционал автоматического добавления слов, который должен включать в себя предоставление пользователю: списка автоматических переводов слова, аудиозапись произношения и транскрипцию.

Для реализации подобного функционала необходимо иметь доступ к базе данных, содержащей множество английских слов и информацию о них. Если учесть, что Оксфордский словарь английского языка содержит 171 476 общеупотребительных слов, а средний активный словарный запас человека, для которого английский язык является родным, составляет 20 000 слов, и пассивный 40 000 слов, то это значит, что подобная база данных должна содержать десятки тысяч слов, еще большее количество переводов их значений, а также транскрипций, аудиозаписей произношений, примеров использования и много всего другого [21].

Создать словарь такого уровня для приложения не представляется возможным в рамках данной выпускной квалификационной работы. А потому было принято прагматичное решение об использовании API открытого словаря, существующего как минимум в двух вариантах, англо-русском и русско-английском.

В рамках данной работы будут рассмотрены два подобных API:

1. Оксфордский словарь.

2. Яндекс словарь.

Оксфордский русско-английский и англо-русский словарь предоставляет открытый интерфейс, который можно использовать для переводов слова с английского на русский и наоборот, и предоставляет следующие типы запросов [33]:

— entries, используются для получения определений, произношений, примеров предложений, грамматической информации и происхождений слов;

— lemmas, используются для проверки, существует ли слово в словаре или с какой «корневой» формой оно связано (например, swimming > swim). Ответ содержит возможные леммы для данного склоняемого слова;

— search, используется для нахождения возможных переводов данного слова;

— translations, используется для того, чтобы вернуть перевод данного слова.

Oxford Dictionaries API предоставляет возможность осуществлять и другие типы запросов, однако они не интересны в рамках данной ВКР.

Dictionaries API предоставляемый компанией Яндекс обладает схожим функционалом, за исключением аудиозаписи произношения слова, что является существенным минусом.

Однако открытый интерфейс Яндекс словаря гораздо проще в использовании, так как предоставляет один многофункциональный метод запроса: lookup, который осуществляет поиск слова или фразы в словаре и возвращает автоматически сформированную словарную статью.

В отличие от обычных переводных словарей, Яндекс Словарь составляется автоматически – с помощью технологий, которые лежат в основе системы машинного перевода Яндекса. В статьях словаря указано, к какой части речи относится слово, а также предоставлены сгруппированные переводы с примерами [26].

Потому для разработки мобильного приложения было принято решение использовать Яндекс Словарь, ввиду простоты его эксплуатации, что позволяет сэкономить немало времени.

Но в таком случае возникает проблема – каким образом должно быть выполнено требование к мобильному приложению о предоставлении пользователю аудиозаписи произношения слова. И в данном случае существует следующее решение. Использовать один из существующих бесплатных и свободных к распространению API.

Один из таких API – Free Dictionary API, который предоставляет десятки тысяч слов английского языка с аудиозаписями их произношений. Использование этого словаря позволит соблюсти все требования к приложению.

3 Разработка мобильного приложения

3.1 Проектирование дизайна мобильного приложения

После определения требований к проекту, его концепции, выбора технологий разработки и проектирования базы данных необходимо перейти к проектированию дизайна мобильного приложения.

При разработке небольших приложений типа MVP [16] – Minimal Viable Product (минимально жизнеспособный продукт), иногда прибегают к интуитивному созданию интерфейса, осуществляемому во время разработки приложения.

Однако во время работы над приложением в рамках данной выпускной квалификационной работы было принято решение о том, что более эффективным вариантом, который поможет качественным образом улучшить эргономичность пользовательского интерфейса, является вынесение создания дизайна в отдельный процесс.

В таком случае потребуется специальное программное обеспечение для выполнения этой задачи и в данном проекте им станет приложение Figma.

Figma – онлайн-сервис для разработки интерфейсов и прототипирования с возможностью организации совместной работы в режиме реального времени [29].

Figma крайне минималистична по дизайну и предоставляет крайне небольшой набор инструментов для разработки макетов, представленный на рисунке 13.

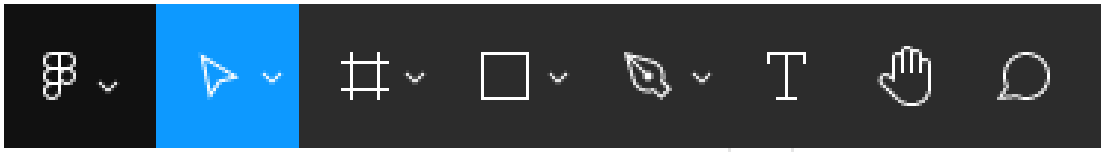

Рисунок 13 – Набор инструментов Figma

Однако этого более чем достаточно для создания дизайнов как простых, так и повышенной сложности.

Разработанный дизайн мобильного приложения для изучения английского языка выглядит так, как это представлено на рисунке 14.

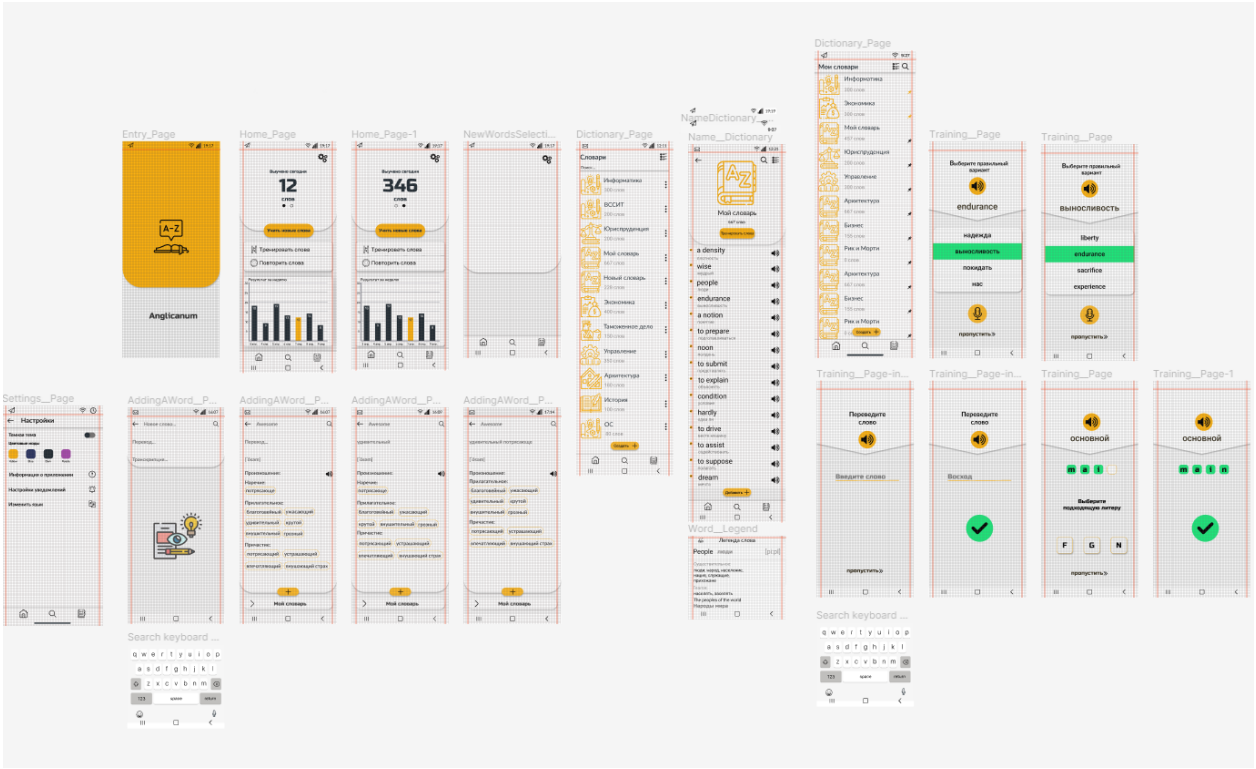

Рисунок 14 – Общий вид на макет дизайна

На представленном выше рисунке располагаются основные элементы дизайна приложения, которые используются в качестве макетов для верстки пользовательского интерфейса.

На следующих рисунках приведен более детальный обзор дизайна экранов приложения.

Сразу после того, как пользователь нажмет на иконку приложения на рабочем столе появится экран загрузки, который называется splash screen, он представлен на рисунке 15.

Рисунок 15 – Splash screen приложения

На рисунке 15 splash screen представлен в двух темах, стандартной белой, а также желтой, выбор цветов интерфейса будет учитываться из предпочтений пользователя.

После того как приложение загрузится, пользователь сразу будет направлен на главный экран.

Главный экран приложения выглядит так, как это представлено на рисунке 16.

Рисунок 16 – Главный экран приложения

Подобно splash screen, главный экран приложения также обладает возможностью персонализации пользовательского интерфейса, как и другие экраны приложения.

В верхней части экрана пользователю показывается основная статистика по использованию приложения. Она включает в себя два пункта, сколько слов пользователь выучил сегодня и при смахе влево сколько слов выучено всего с момента начала использования приложения.

Далее находится кнопка желтого цвета с надписью учить слова, при нажатии на нее будет открываться окно, в котором пользователю будут рекомендоваться новые слова для изучения.

Еще ниже располагаются две кнопки:

1. Тренировать слова.

2. Повторить слова.

Кнопка «Повторить слова» нужна для повторения уже изученных слов, а кнопка «Тренировать слова» для перехода к экрану, на котором пользователю будут предлагаться тренировки для лучшего запоминания слов.

Также пользователю будет предлагаться статистика изучения слов в виде графика за текущую неделю, как это представлено на рисунке 17.

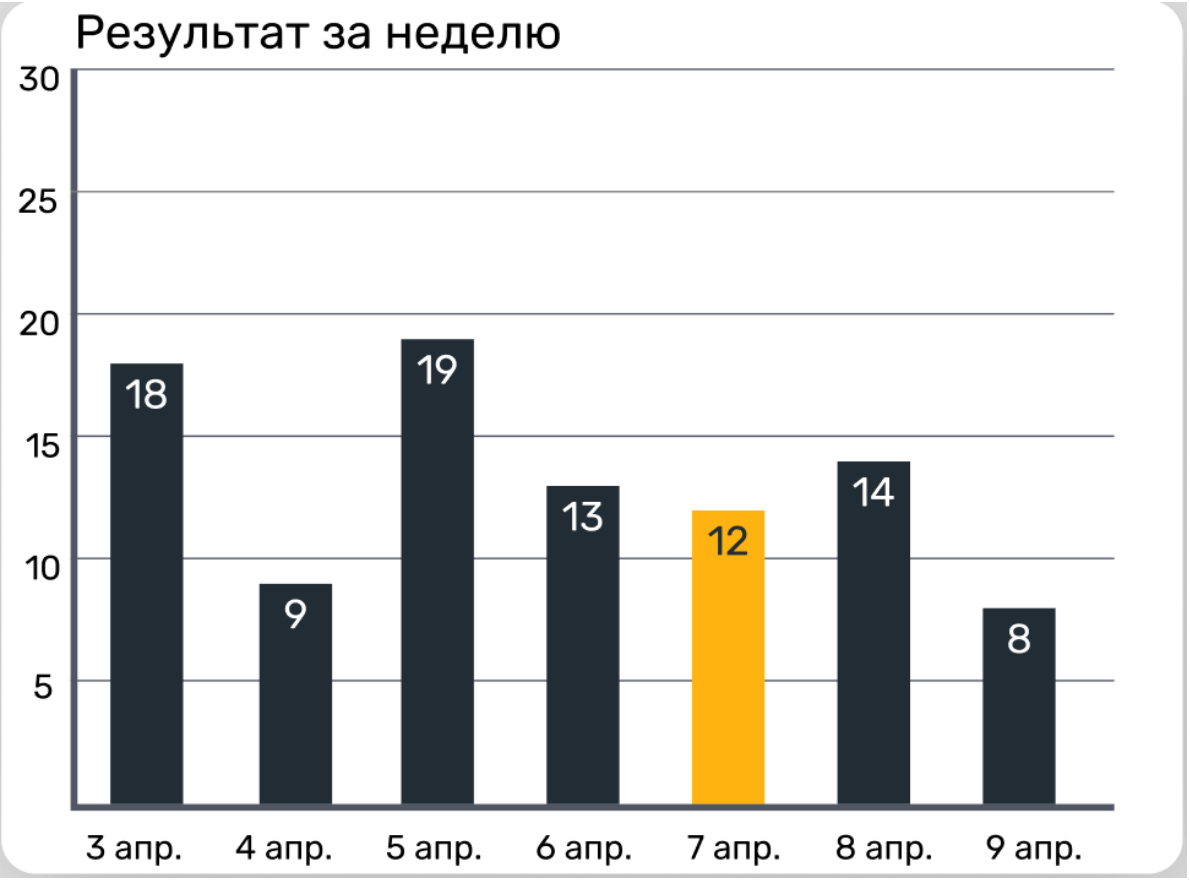

Рисунок 17 – График со статистикой изучения слов

Также в верхнем правом углу располагается кнопка в виде шестеренок, при нажатии на нее пользователь переходит на экран настроек приложения, представленный на рисунке 18.

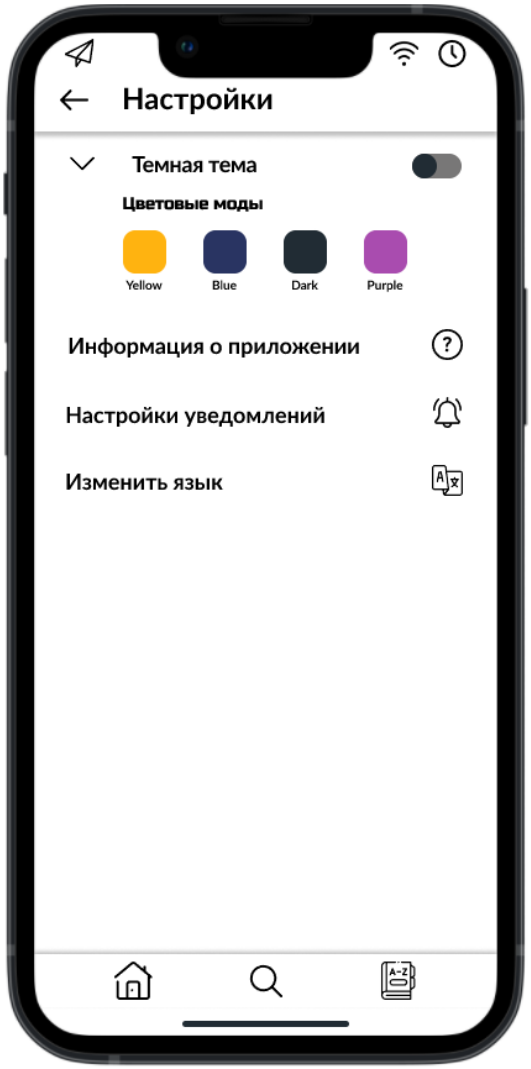

Рисунок 18 – Экран настроек приложения

В окне настроек можно выполнить следующие действия:

— изменить тему приложения, она представлена в двух популярных вариантах: светлая и темная;

— выбрать цветовой мод для приложения, он будет использоваться для стилистического оформления некоторых элементов интерфейса приложения;

— получить информацию о приложении, в том числе информацию о разработчике и способах связи с ним;

— настроить нотификацию;

— изменить язык интерфейса, он представлен в двух вариантах: русский и английский, это сделано для лучшего погружения в изучение языка.

Также стоит рассмотреть макеты дизайна экрана тренировки слов. Концептуально тренировка будет представлена тремя вариантами:

1. Выбор правильного слова, как на русском, так и на английском языке, с возможностью голосового ввода.

2. Ручной ввод перевода слова.

3. Побуквенный ввод слова.

На рисунке 19 представлен первый вариант тренировки.

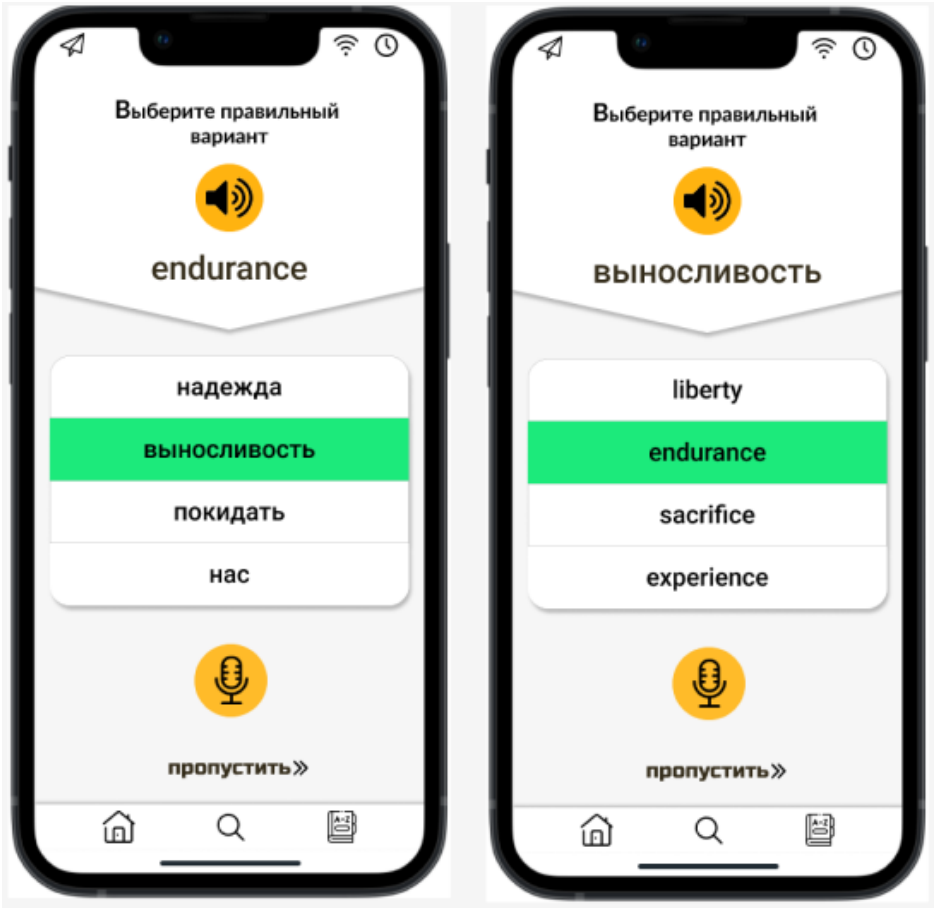

Рисунок 19 – Первый вариант тренировки

Концепция тренировок состоит в том, чтобы пользователь мог учить новые слова в игровом формате, так как это способствует лучшему запоминанию и позволяет с интересом выучить больше новых слов.

На рисунке 20 представлен второй вариант тренировки.

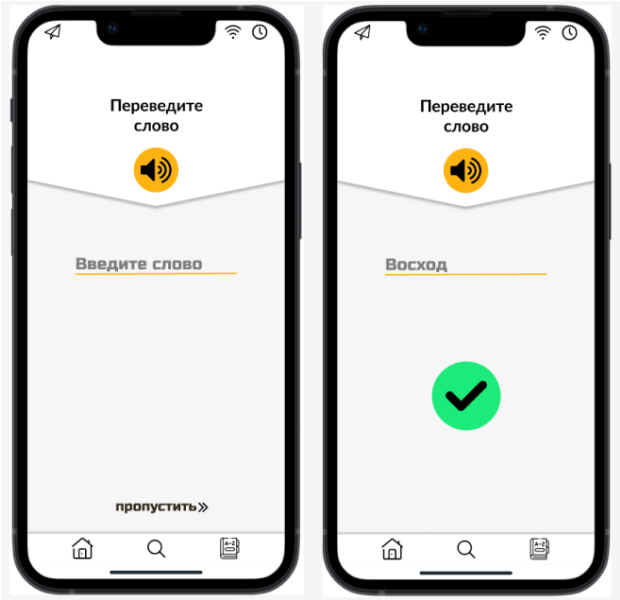

Рисунок 20 – Второй вариант тренировки

И на рисунке 21 представлен третий вариант тренировки.

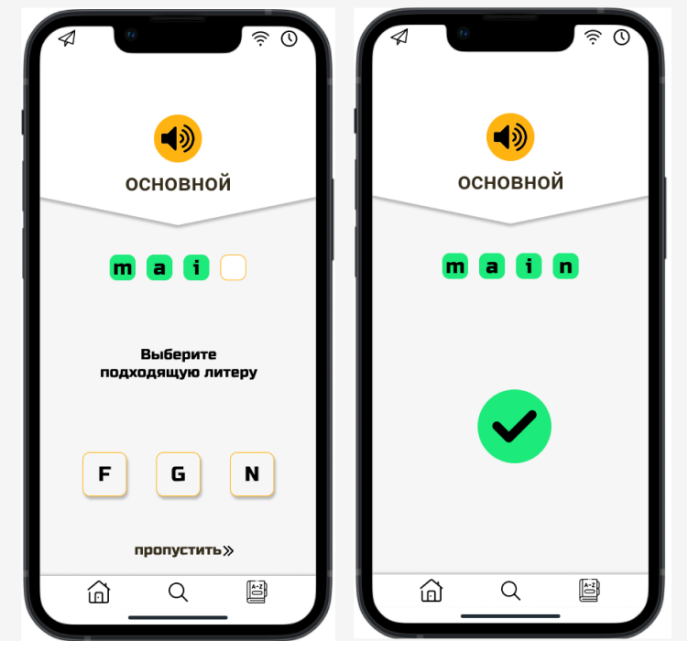

Рисунок 21 – Третий вариант тренировки

На этом этапе пользователю предлагается заполнять перевод слова по буквам, выбирая из трех вариантов.

3.2 Конструирование мобильного приложения

После того как ключевые макеты дизайна приложения разработаны можно приступать к разработке самого приложения.

Для тестирования работы приложения использовались несколько разных эмуляторов Android, для удостоверения в работоспособности приложения.

Директория, содержащая файлы с исходным кодом, выглядит как это показано на рисунке 22.

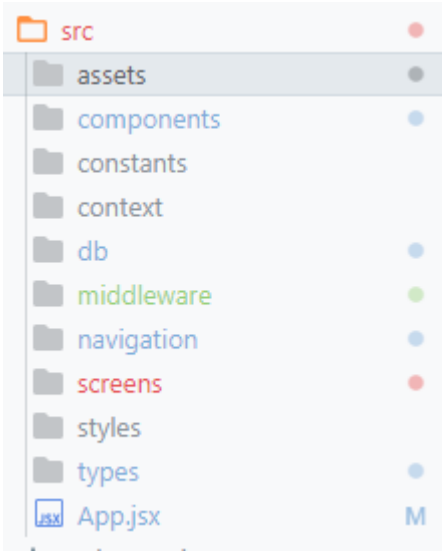

Рисунок 22 – Директория исходного кода

Содержащиеся в ней каталоги представляют собой следующее:

— assets, содержит файлы изображения и иконок используемых в приложении;

— components, здесь хранятся фалы с основными переиспользуемыми компонентами приложения;

— constants, содержит перечень констант для приложения, таких как, например, url-адрес открытого API словаря;

— context, используется для создания контекста внутри приложения;

— db, инициализация и использование базы данных;

— middleware, содержит файлы с кодом для обработки сетевых запросов;

— navigation, здесь хранятся файлы для создания навигации по экранам внутри приложения;

— screens, хранит компоненты представляющие собой экраны приложения;

— styles, хранит некоторые стили компонентов;

— types, так как приложение использует технологию строгой статической типизации «Flow», некоторые типы используемые в приложении повсеместно, вынесены в этот каталог;

— App.js, основной файл приложения, его код представлен в приложении А.

Когда пользователь запускает приложение он попадает на главный экран, который представлен компонентом «Home», часть кода которого изображена на рисунке 23.

```
type HomeProps = {
    navigation: any,
    theme?: ColorSchema
};

export default class Home extends Component<HomeProps> {

    static contextType: ThemeContextType = ThemeContext;

    render(): Node {

        const theme: ColorSchema = this.context.theme;

        return (
            <View style={[styles.home, {backgroundColor: theme?.background}]}>
                <HomeView
                    theme={theme}
                    navigation={this.props.navigation}
                />
                <CustomBtn
                    style={styles.homeViewBtn}
                    styleTxt={styles.homeViewBtnText}
                    text='Учить слова'
                />
                <View style={styles.homeMain}>
                    <TrainButtons
                        theme={theme}
                        navigation={this.props.navigation}
                        />
                </View>
            </View>
        );
    }
}
```

Рисунок 23 – Код компонента Home

В данном варианте этот компонент представляет из себя мало интересного, однако в верхней его части располагается объявление типа HomeProps, этот тип представляет то, какие свойства может принять данный компонент. При объявлении класса тип передается в дженерик класс Components, от которого наследуется компонент Home.

Также в первой строке класса можно увидеть переопределяемую статическую переменную contextType, как это видно на рисунке 24.

```
static contextType: ThemeContextType = ThemeContext;
```

Рисунок 24 – Переопределение contextType

Это требуется для использования контекста, который нужен для динамической персонализации пользовательского интерфейса.

Исходный код экрана настроек, доступный только из главного экрана и макет дизайна которого был изображен на рисунке 18 представлен в приложении Б.

Одной из самых важных частей приложения является навигационный бар, он содержит три кнопки для переходов по следующим экранам:

1. Главный экран.
2. Экран поиска слов.
3. Словари.

Навигационный бар выглядит как это представлено на рисунке 25.

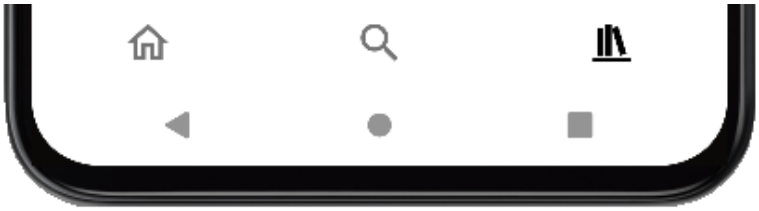

Рисунок 25 – Навигационный бар

Основной код навигационного бара представлен в приложении В.

Как видно из рисунка 25, каждая из трех иконок бара подсвечивается в зависимости от того на каком экране в данным момент находится пользователь. Это осуществляется при помощи кода, представленного на рисунке 26.

```
function setIcon(route: any, size: number): (props: MenuIconProps) => Node {
    return function MenuIcon({color}: MenuIconProps): Node {
        let iconName: MaterialCommunityIconsGlyphs = 'home-outline';
        switch(route.name) {
            case 'Search':
                iconName = 'magnify'
            break;
            case 'Dict':
                iconName = 'bookshelf'
            break;
        }
        return <Icon name={iconName} size={size} color={color} />;
    }
}
```

Рисунок 26 – Код для настройки иконок

На рисунке 26 изображена функция setIcon, которая возвращает замыкание, представляющее собой компонент для отрисовки иконки, MenuIcon. Логика этого компонента позволяет динамически отрисовывать иконку в зависимости от имени используемого маршрута навигации.

Теперь наконец можно перейти к самой важной части приложения, отвечающей за создание словарей, а также добавление и хранение слов.

На рисунке 27 представлен вид экрана эмулятора, на котором изображено окно словарей.

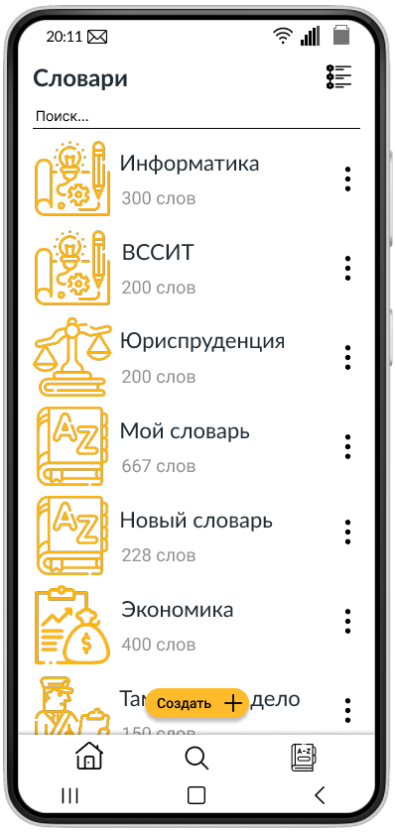

Рисунок 27 – Экран – Словари

На рисунке 27 представлен экран содержащий в себе список словарей, некоторые из которых созданы пользователем, а другие, такие как, «Информатика» или «Экономика», предоставлены приложением и содержат перечень слов полезных студентам, обучающимся на этой специальности.

В нижней части экрана расположена кнопка «Создать», при нажатии на нее всплывает модальное окно для ввода имени нового словаря, которое продемонстрировано на рисунке 28.

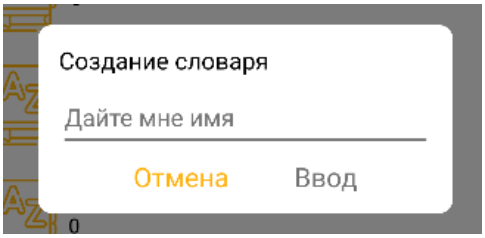

Рисунок 28 – Модальное окно

Общий вид на создание нового словаря представлен на рисунке 29.

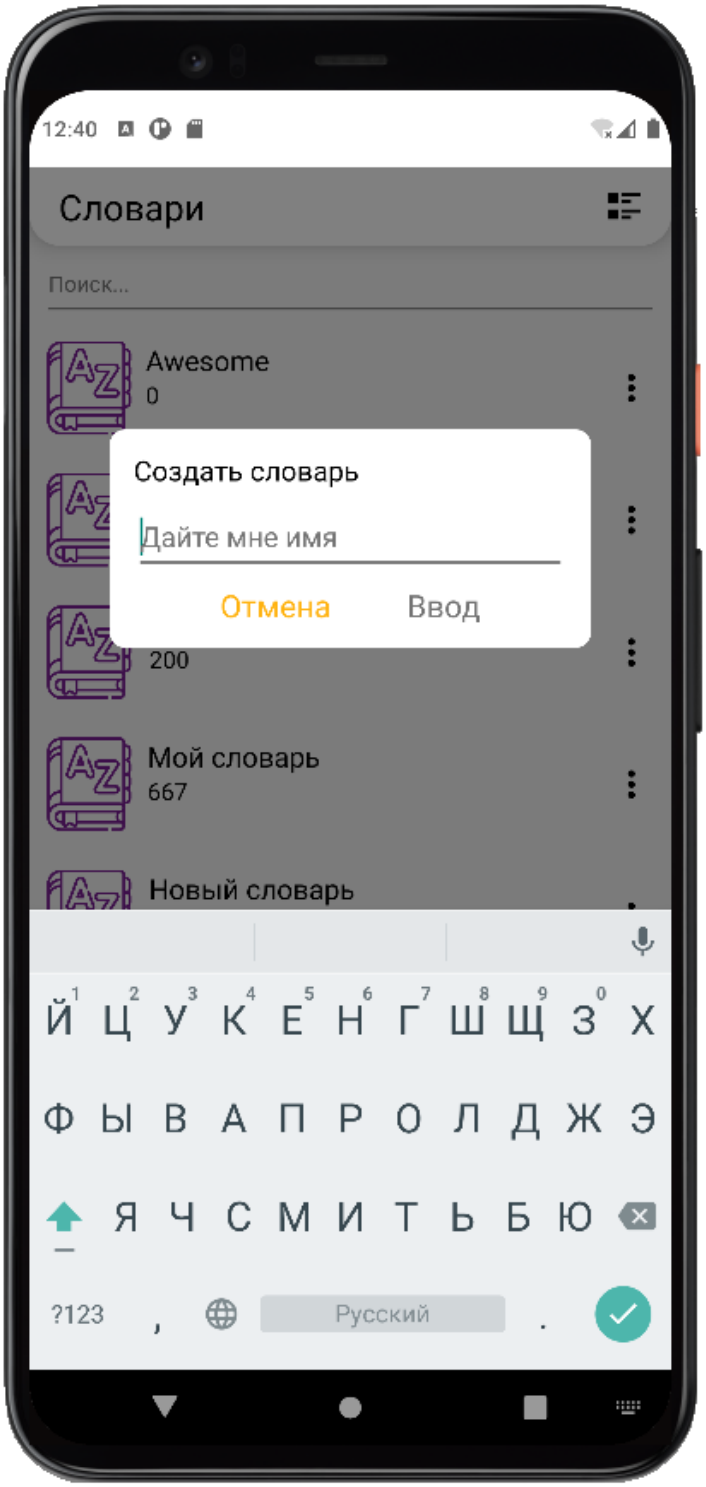

Рисунок 29 – Темная тема

За открытие модального окна отвечает метод компонента Dictionaries, изображенный на рисунке 30.

```
handleModalVisibility: Function = function(): void {
    this.setState(preventState => ({
        modalIsOpen: !preventState.modalIsOpen
    }));
};
```

Рисунок 30 – Метод для открытия модального окна

За создание и удаление словарей в базе данных отвечают методы: handleAddDict и handleDeleteDict, представленные на рисунке 31.

```
handleAddDict: Function = function(name: string): void {
    realm.write(() => {
        realm.create('Dict', new Dict({
            name,
            wordsAmount: this.state.words.length,
            icon: 'purpleDictionary',
            words: this.state.words
        }));
    });
};

handleDeleteDict: Function = function(dict: DictType): void {
    realm.write(() => {
        realm.delete(dict);
    });
    this.handleUpdateDicts();
};
```

Рисунок 31 – Методы создания и удаления словаря

Метод handleAddDict получает на вход имя создаваемого словаря и затем совершает запрос к базе данных на добавление записи, задает имя, количество слов и иконку словаря.

Метод handleDeleteDict удаляет словарь и обновляет список показываемых пользователю словарей.

Для того чтобы удалить словарь, необходимо сделать смах влево, как это показано на рисунке 32 и тогда появиться кнопка удаления.

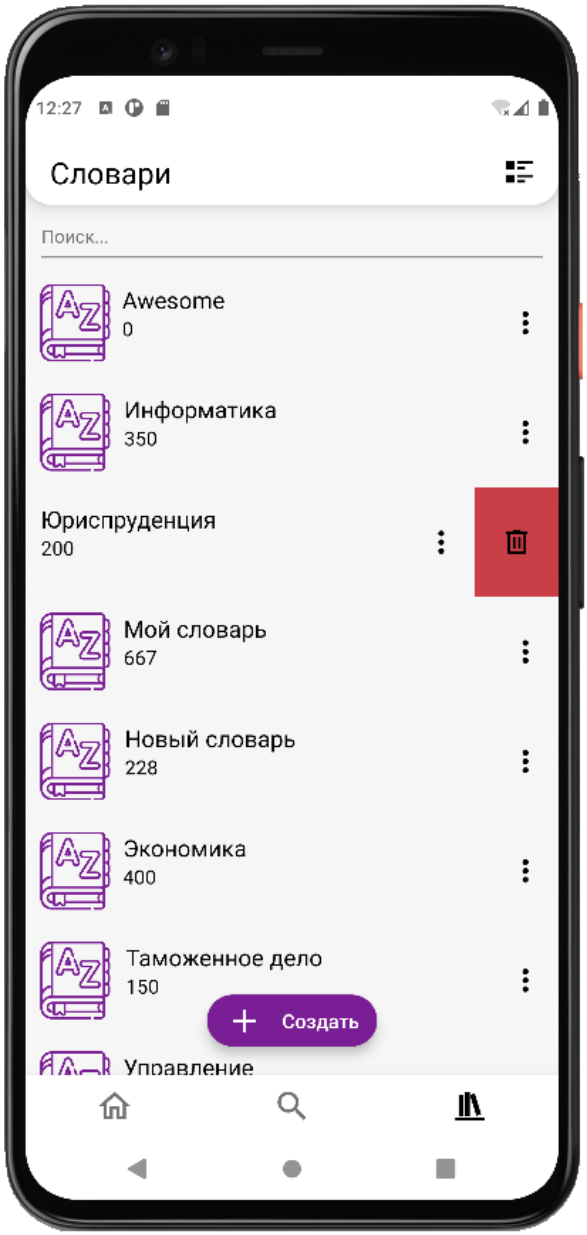

Рисунок 32 – Удаление словаря

Также пользователь может хранить большое количество словарей, в таком случае ему может понадобиться поиск по списку. И потому на странице

словарей находится строка поиска, она представлена в верхней части списка словарей на рисунках: 27, 29, 32.

Можно предположить следующее, пользователь хочет найти словарь с именем «Awesome», это может быть осуществлено так, как это показано на рисунке 33.

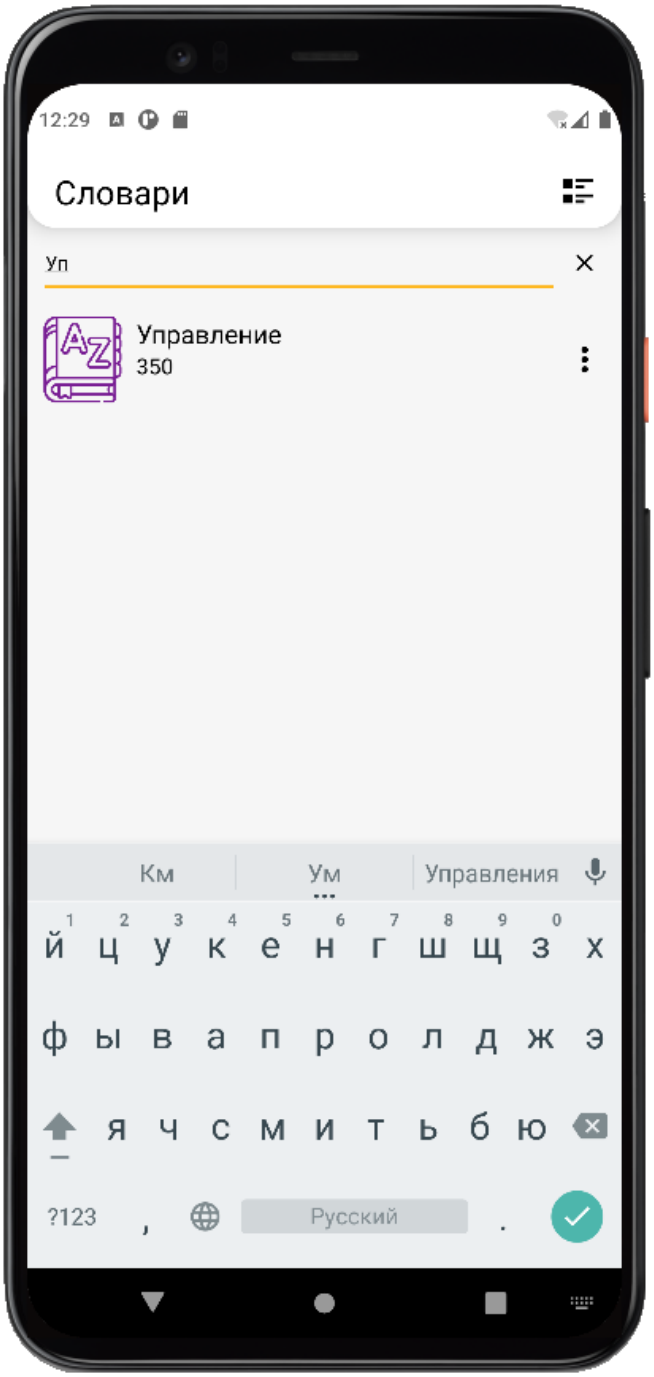

Рисунок 33 – Поиск по словарям

Достаточно набрать первые символы названия, как приложение уже формирует список подходящих словарей.

Полный код компонента Dictionaries, отвечающего за отображение и бизнес-логику экрана «Словари», представлен в приложении Г.

Каждый словарь содержит список слов, определенный пользователем. Чтобы открыть окно с этим списком конкретного словаря, нужно на него нажать. И тогда откроется окно, представленное на рисунке 34.

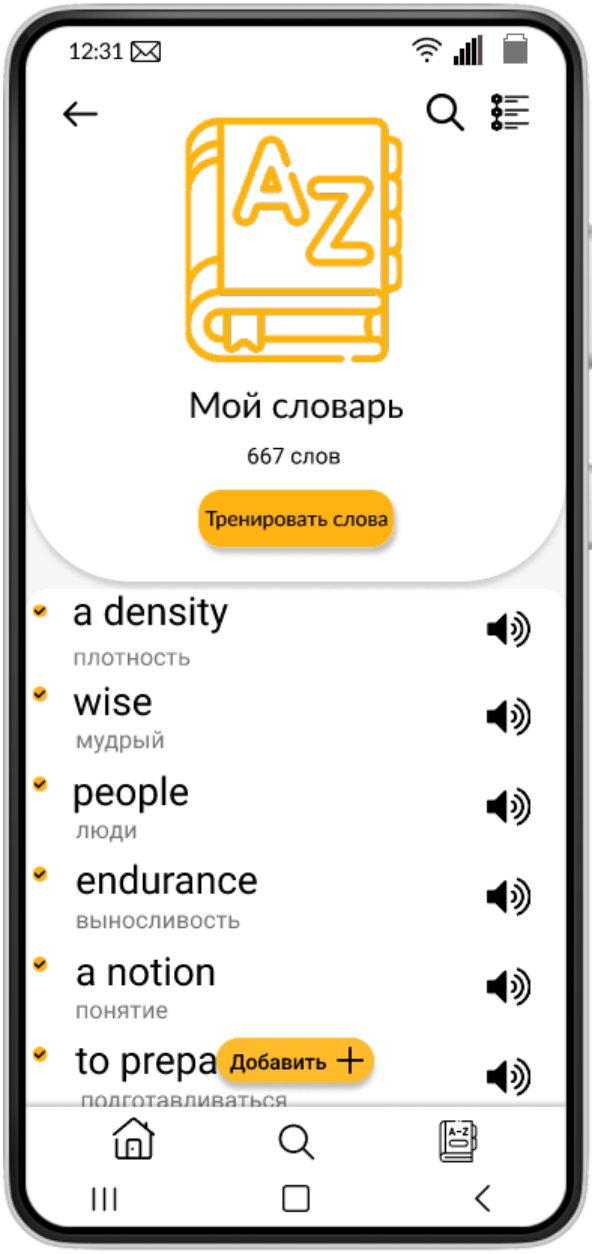

Рисунок 34 – Окно со списком слов

В верхней части окна располагаются три кнопки:

1. Кнопка назад.

2. Поиск по словам.

3. Сортировка слов.

В списке, напротив каждого слова расположена иконка мегафона, при нажатии на нее пользователь услышит аудиозапись произношения слова.

При нажатии на само слово всплывет модальное окно, которое будет содержать всю информацию о слове, хранимую в базе данных, как это представлено на рисунке 35.

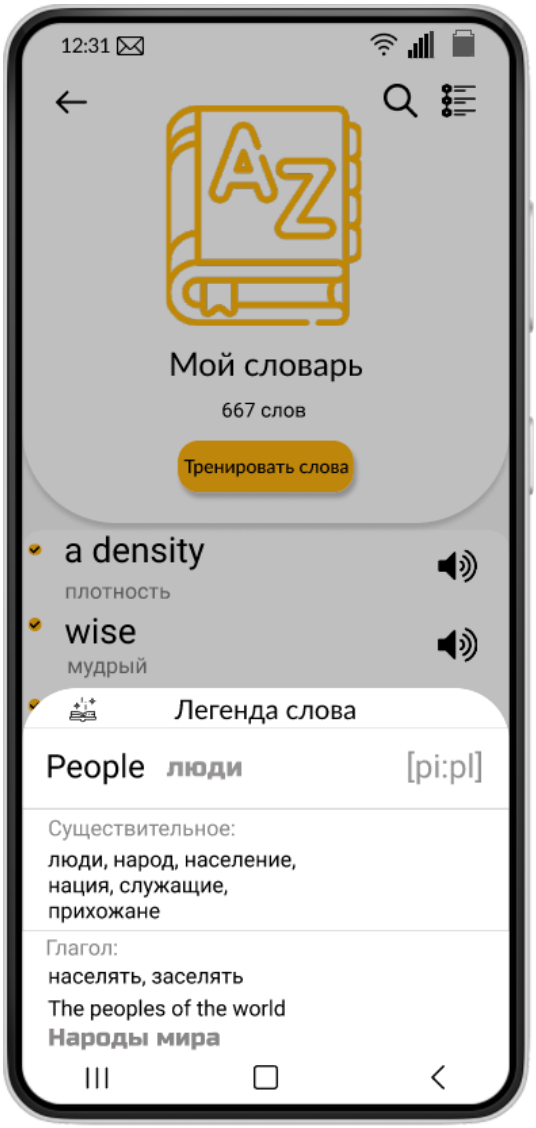

Рисунок 35 – Модальное окно с информацией о слове

В нижней части окна на рисунке 34, располагается кнопка «Добавить +», при нажатии на нее пользователь переходит на экран добавления новых слов. На этом экране пользователь может ввести слово на английском или на русском языке, ввести его перевод и транскрипцию.

Как было описано во второй главе, в пункте, посвященном требованиям к ПО, приложение также предоставляет полный набор переводов слова группированным по частям речи, также транскрипцию и аудиозапись произношения.

Окно добавления новых слов представлено на рисунке 36.

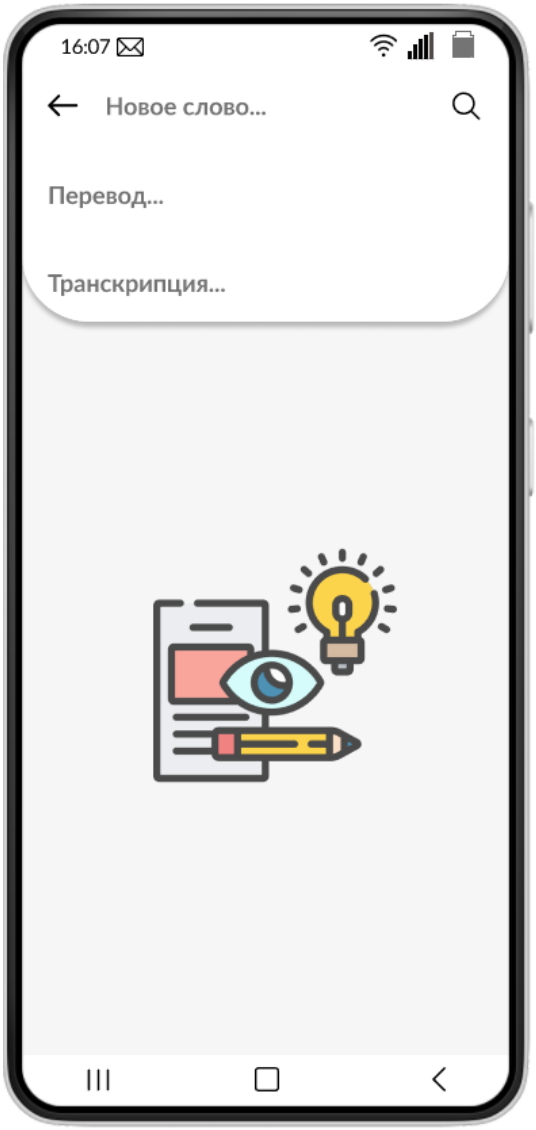

Рисунок 36 – Окно добавления новых слов

После ввода слова и нажатия на кнопку поиска загружается информация как это представлено на рисунке 37.

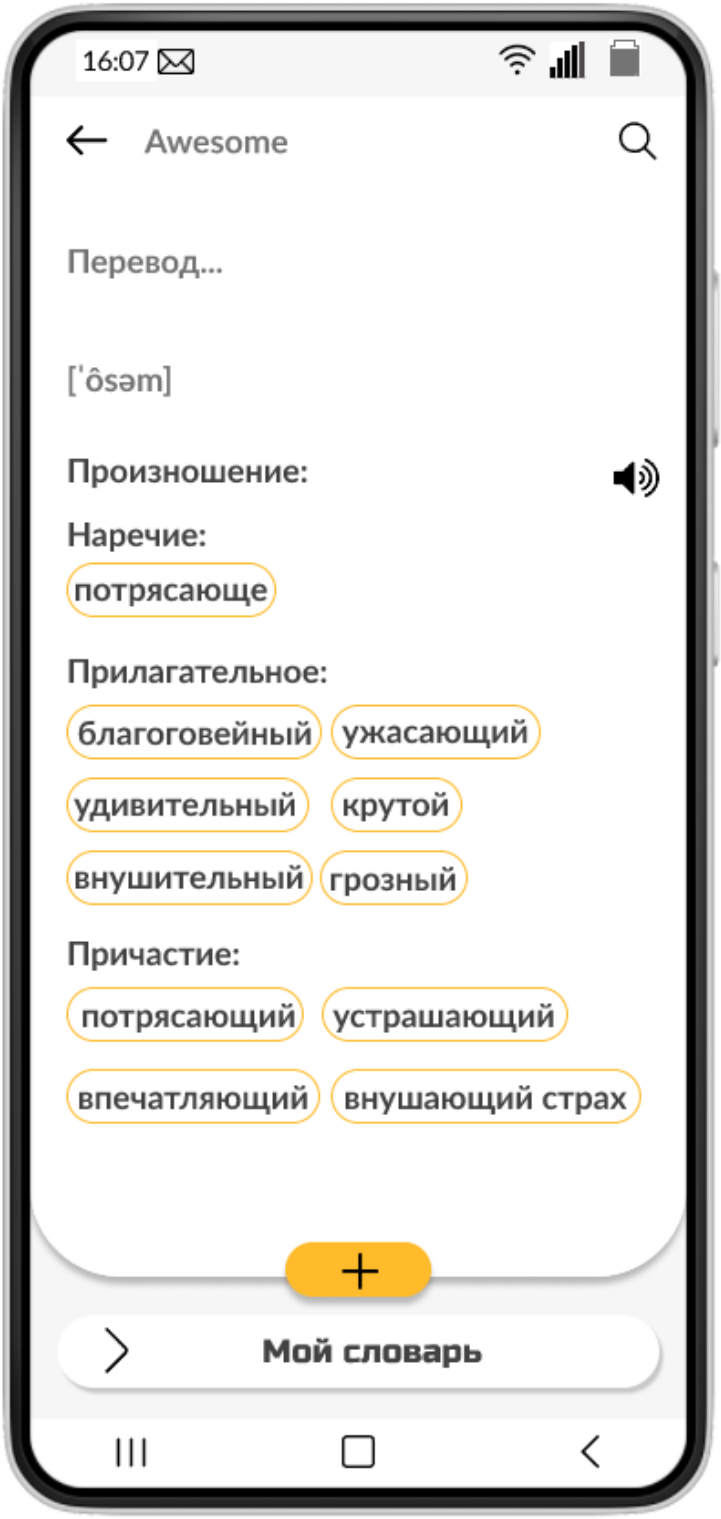

Рисунок 37 – Загруженная информация о слове

За этот функционал отвечает метод handleSearchWord компонента AddingAword, представленный на рисунке 38.

```
handleSearchWord: Function = function(): void {
    if (this.state.word !== '') {
        const lang: string = defineLang(this.state.word);

        searchWord(this.state.word, lang)
            .then(word => {
                if (word ?? false) {
                    this.setState({
                        wordDef: word,
                        sound: this.setSound(
                            word.pronunciation
                        ),
                        transcription:
                            word.transcription ?? false ?
                                `[${String(word.transcription)}]`
                                : ''
                    });
                }
                Keyboard.dismiss();
                return;
            })
            .catch((e: Error) =>
                console.log('search', e.message)
            );
    } else {
        alert('Поле запроса не должно быть пустым');
    }
};
```

Рисунок 38 – Метод для загрузки информации о слове

Для того чтобы задать слову варианты перевода, нужно нажимать на кнопки из автоматически предоставленных переводов, и они будут добавляться в строфу «Перевод», как это изображено на рисунке 39.

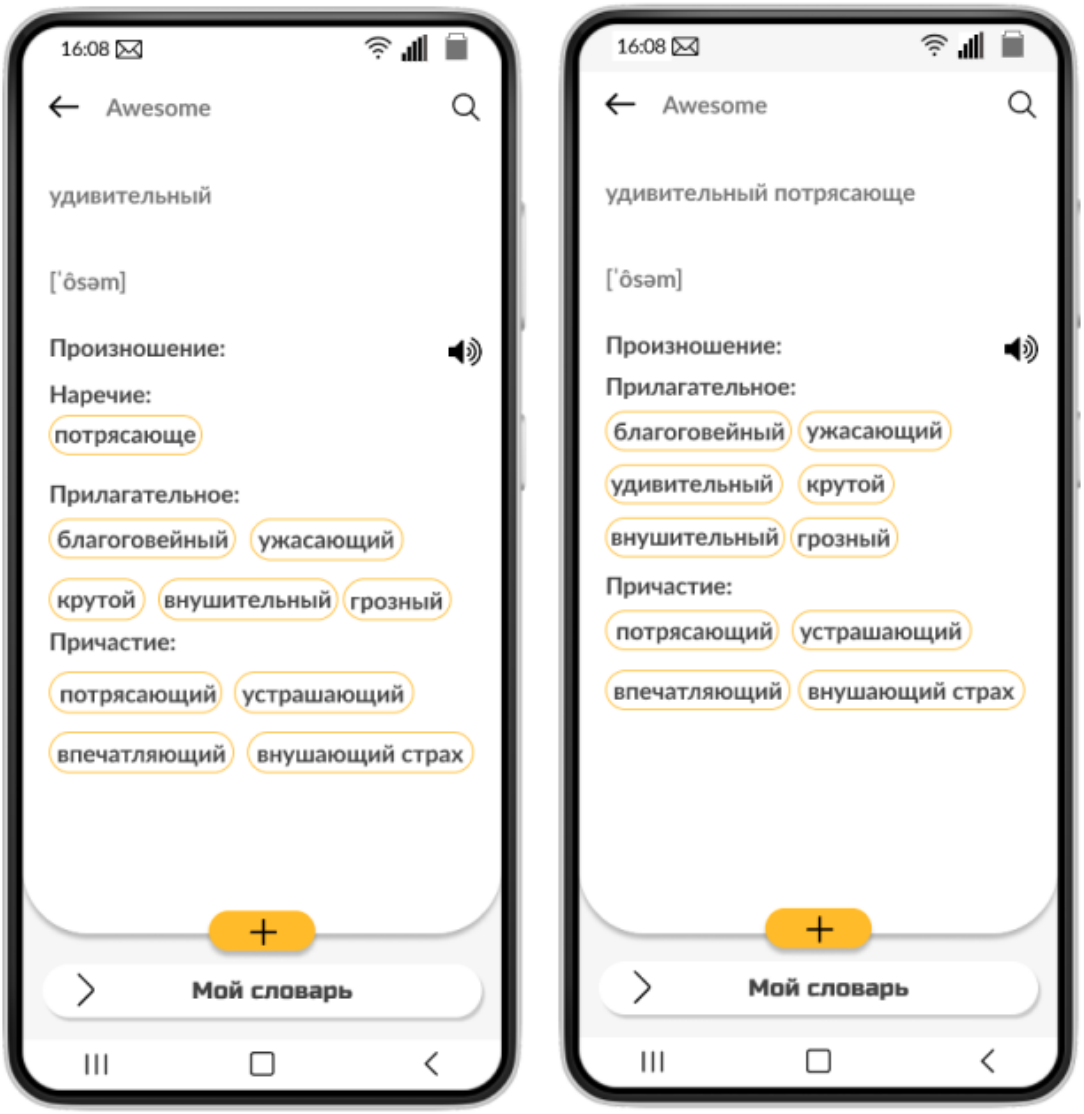

Рисунок 39 – Добавление переводов

При нажатии на кнопку с плюсом слово будет добавляться в тот словарь, который указан пользователем в меню выбора, расположенном ниже этой кнопки.

За это отвечает компонент AddingAWord, полный код которого представлен в приложении Д.

За автоматический перевод отвечает модуль dictApi. Определенная в нем функция searchWord, основной код которой представлен на рисунке 40, осуществляет два асинхронных запроса с помощью библиотеки Axios к API

словарей и получает два json документа, один из которых содержит набор переводов значений, а другой набор аудиозаписей произношений слова.

```
try {
    const resFromYndxApi= await axios.get(
        yandexApiUrl.concat(yandexApiMethod.lookup),
        {
            params: {
                key: yandexApiKey,
                text,
                lang,
                ui: 'ru'
            }
        }
    );
    fillWord(resFromYndxApi.data, word);

    if (lang === 'en-ru') {
        const resFromFreeApi = await axios.get(
            freeApiUrl.concat(text)
        );
        fillPronun(resFromFreeApi.data, word);
    }

    return word;

} catch (error) {
    console.log('http', error.message);
}

return {};
```

Рисунок 40 – Часть кода функции searchWord

После запросов эта функция осуществляет вызов функции fillWord и fillPronun. Первая из них парсит переводы значения слова, а вторая аудиозаписи произношений. Код этих функций приведен на рисунке 41.

```
import type { WordType } from '../types';
import axios from 'axios';

function fillWord(wordDef: Object, word: WordType): void {
    word.name = wordDef.def[0].text;
    word.transcription = wordDef.def[0].ts;

    wordDef.def.forEach(def => {
        def.tr.forEach(tr => {
            word.partsOfSpeech?.get(
                def.pos
            )?.push(tr.text);
        });
    });
}

function fillPronun(wordPronun: Object, word: WordType): void {
    wordPronun[0].phonetics.forEach(phonetic => {
        if (phonetic.audio !== '') {
            word.pronunciation = phonetic.audio;
        }
    });
}
```

Рисунок 41 – Код функций для парсинга значений

4 Расчет экономической эффективности приложения

4.1 План выполненных работ

Для того чтобы оценить преимущества от ввода разработанного ПО в эксплуатацию, необходимо проанализировать тот экономический эффект и его полезность для предприятия, который приложение может дать.

Потому необходимо постфактум оценить количество времени, потраченное на выполнение задач по проекту и их трудоемкость.

Календарный план выполненных задач представлен в таблице 1.

Таблица 1 – Календарный план выполнения задач

№	Выполненная задача	Время (в днях)	Начало выполнения задачи	Конец выполнения задачи
1	Выявление проблемы	1	27.03.2022	27.03.2022
2	Разработка решения	1	28.03.2022	28.03.2022
3	Определение требований к разработке	2	29.03.2022	30.03.2022
4	Определение требований к приложению	2	31.03.2022	01.04.2022
5	Проектирование концепции дизайна	10	03.04.2022	12.04.2022
6	Подготовка среды разработки	4	14.04.2022	17.04.2022
7	Проектирование мобильного приложения	7	19.04.2022	25.04.2022
8	Проектирование базы данных	3	27.04.2022	29.04.2022
9	Разработка интерфейса пользователя	12	30.04.2022	11.05.2022
10	Конструирование мобильного приложения	30	12.05.2022	10.06.2022
11	Подключение к API открытого словаря	1	11.06.2022	11.06.2022
12	Отладка и тестирование функционала	6	12.06.2022	17.06.2022

Как видно из таблицы 1, количество выполненных задач равно двенадцати, а совокупное время, потраченное на их исполнение равно семидесяти девяти дням.

При средней зарплате младшего мобильного разработчика в Хабаровске 40 000 рублей/месяц и пятидневной рабочей недели, можно рассчитать примерную стоимость проекта.

Если считать среднюю длину месяца равной 30 дням, а количество выходных равным 8, то разработчик получает 40 / (30 - 8) = 1.81 тысячи рублей в сутки, таким образом общая примерная стоимость проекта равна 79 1.81 = 142.99 тысячи рублей.

При затратах на разработку не стоит учитывать расходы на программное обеспечение, по причине того, что все технологии использовавшиеся при разработке ПО распространяются бесплатно.

4.2 Экономическая эффективность в контексте приложения

Понятие экономической эффективности может быть выражено в следующем определении [24]:

Экономическая эффективность – это величина, определяемая соотношением полученных результатов деятельности человека, производства продукции (товаров или услуг) и затрат труда и средств на производство.

Мобильное приложение, разрабатываемое в рамках данной выпускной квалификационной работы ориентированно на использование студентами ФГБОУ ВО «ХГУЭП», и, следовательно, не подразумевает наличия коммерческой составляющей.

Основная цель приложения – принесение практической пользы для обучающихся университета и косвенным образом для их альма-матер.

Практическая польза от разрабатываемого ПО представляет собой избавление от следующих издержек:

— студенты вынуждены тратить много времени на изучение и запоминание слов, что негативно сказывается на процессе их обучения;

— многие студенты пользуются платными аналогами разрабатываемого приложения, которые чаще всего не выгодны по цене.

Экономическая эффективность рассчитывается по формуле:

$$Э = Э_р - Е_н * К_{пр}\ , \qquad (1)$$

где $Э_р$ – годовая экономия;

$Е_н$ – нормативный коэффициент ($Е_н$ = 0,3);

$К_{пр}$ – вложения на проектирование и разработку.

Снижение издержек достигается исходя из экономии на расходы по эксплуатации и экономии времени ввиду увеличения эффективности работы пользователя. Таким образом, получаем:

$$Э_р = (Р1 - Р2) + \Delta Р_п, \qquad (2)$$

где Р1 и Р2 – эксплуатационные расходы до и после внедрения разрабатываемого ПО;

$\Delta Р_п$ – экономия от увеличения эффективности и качества работы пользователей.

Издержки обучающегося и университета как до, так и после внедрения программы остаются неизменными. Ввиду этого годовая экономия будет равна экономии, связанной с повышением эффективности труда пользователя мобильного приложения.

Если пользователь при выполнении работы i с применением программы тратит ΔT_iвремени, то увеличение эффективности труда определяется по формуле 3:

$$P_i = \left(\frac{F_j - \Delta T_j}{\Delta T_j}\right) * 100\%, \qquad (3)$$

где F_j – время, которое планировалось пользователем для выполнения работы j до введения ПО в эксплуатацию.

Прежде чем вычислить повышение эффективности труда пользователя, необходимо явно обозначить то, на что пользователь обычно расходует время для решения своих задач:

— поиск нового слова; Разработанное мобильное приложение поможет пользователю сократить время на поиск благодаря тому, что имеет возможность переводить слова не только с английского на русский, но и с русского на английский, и если пользователь хочет узнать перевод, он сможет сделать это предельно быстро и сразу же сохранить результат;

— получение всех возможных переводов слова, транскрипции и варианта произношения. До внедрения приложения пользователю приходилось делать это все вручную, но теперь этот процесс происходит автоматически. Приложение дает все возможные переводы, группируя их по частям речи, также предоставляет транскрипцию и запись произношения, что экономит много времени;

— внесение всех данных в словарь для запоминания. До внедрения ПО, чтобы запомнить слово, пользователю необходимо было вручную вести учет всех изучаемых слов, однако, приложение предоставляет возможность создания персонализированных словарей, автоматизировав тем самым и этот процесс.

В таблице 2 представлены виды задач пользователя по изучению нового слова.

Таблица 2 – Задачи пользователя

Вид работы	До автоматизации, мин. F_j	После автоматизации. ΔT_j	Повышение эффективности труда, %
1	2	3	4
Поиск нового слова	1 минута	Менее 1 минуты, 0.5	100

Продолжение таблицы 2

1	2	3	4
Получение переводов слова, транскрипции и варианта произношения	2.5 минут	1 минута	150
Внесение данных в словарь для запоминания	4 минуты	1 минута	300

Исходя из данных представленных в таблице 2, можно рассчитать, что среднее повышение эффективности равно 200%. Тогда можно вычислить в часах годовую экономию времени от внедрения приложения по следующей формуле:

$$f = \frac{Q * T\text{слово} * T\text{дней}}{60}, \quad (4)$$

где f – затраченное количество времени в год;

Q – количество изучаемых слов;

$T_{\text{слово}}$ – время на одно слово;

$T_{\text{дней}}$ – количество дней в год.

Если пользователь желает изучать 10 новых слов в сутки, 5 дней в неделю на протяжении года (то есть примерно 52 недели), то в таком случае, согласно формуле 4 и данным из таблицы 2, примерно 10 * 7.5 * (5 * 52) / 60 = 325 часов в год будет расходоваться на изучение слов до ввода приложения в эксплуатацию. А после ввода 10 * 2.5 * (5 * 52) / 60 = 108.33 часа в год, а это означает, что пользователь будет экономить 325–108.33 = 216.67 часов ежегодно.

4.3 SWOT-анализ

Для осуществления стратегического планирования с целью оценить перспективность разработки мобильного приложения, рационально будет составить матрицу SWOT-анализа.

Посредством данной матрицы представляется возможность оценить сильные и слабые стороны реализуемого проекта, а также возможности и потенциальные риски, как это представлено в таблице 3.

Таблица 3 – Матрица SWOT-анализа

	Возможности: Перспектива масштабирования возможностей продукта. Увеличение спроса на предоставляемые продуктом услуги.	Угрозы: Наличие большого количества конкурентов.
Сильные стороны: Наличие узкоспециализированного функционала. Эргономичность пользовательского интерфейса. Кроссплатформенность.	Масштабирование продукта в сторону предоставления большего функционала для разных групп потенциальных клиентов. Рекламирование системы среди пользователей.	Ставка на более гибкую цену и удобство пользовательского интерфейса с целью удержание интереса клиентов к продукту.

Продолжение таблицы 3

1	2	3
Слабые стороны: Небольшой по объему функционал. Невысокая производительность. Низкая отказоустойчивость системы.	Расширение предоставляемого функционала. Использование новых технологий для совершенствования системы. Привлечение новых специалистов для решения технических проблем.	Высокая вероятность уменьшения конкурентоспособности. Вероятность потери клиентов.

Как результат проведения SWOT-анализа были выявлены сильные и слабые стороны, а также возможности и угрозы, в результате чего стало ясно на какие детали прежде всего стоит обратить внимание в случае дальнейшего масштабирования приложения.

ЗАКЛЮЧЕНИЕ

Как результат выполнения данной выпускной квалификационной работы было спроектировано и разработано мобильное приложение для изучения английского языка студентами ФГБОУ ВО «ХГУЭП».

Этот процесс был осуществлен в несколько этапов:

1. Исследование предметной области.
2. Постановка и анализ проблемы.
3. Разработка решения.
4. Определение концепции приложения.
5. Анализ существующих решений.
6. Выбор технологического стэка.
7. Формулирование требований к разработке.
8. Формулирование требований к ПО.
9. Подготовка рабочего пространства.
10. Разработка дизайна приложения.
11. Конструирование приложения.
12. Тестирование и отладка.
13. Анализ экономической эффективности.

На этапе исследования предметной области была выявлена и проанализирована проблема и представлен вариант ее решения. Затем были определенны все концептуальные особенности приложения.

Также были рассмотрены существующие решения данной проблемы, вывялены их недостатки и предложена концепция, покрывающая их недостатки и предлагающая дополнительные преимущества.

Далее на основе статистических данных было обозначено для каких платформ будет создаваться ПО и проведен анализ существующих на рынке инструментов для разработки мобильного приложения под эти платформы. Это позволило определить подходящий технологический стэк, который в последствии был применен в конструировании ПО.

Помимо этого, сформулированы требования к разработке ПО и к мобильному приложению.

Во время подготовки рабочего пространства была описана структура базы данных и зависимости разрабатываемого приложения. Рассмотрен API открытых словарей и обоснован выбор одного из них для использования приложением.

На этапе разработки дизайна были изготовлены макеты основных компонентов пользовательского интерфейса.

Следующим этапом стало конструирование ПО, где был представлен основной функционал мобильного приложения. Также был приведен код отвечающий за работку основных компонентов и описан принцип работы приложения.

На последнем этапе был произведен расчет экономической эффективности при введении мобильного приложения в эксплуатацию. Расчеты показали, что при внедрении приложения, обучающиеся университета получат трехкратную экономию времени обучения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования : пер. с англ. СПБ. : Питер, 2016. 368 с.

2 Дежина Т.П. Рабочая программа дисциплины – Иностранный язык. Хабаровск, 2021. 22 с.

3 Макконнелл С. Совершенный код. Мастер-класс : пер. с англ. СПБ. : БХВ, 2021. 896 с.

4 Симпсон, К. ES6 и не только : пер. с англ СПБ. : Питер, 2017. 336 с.

5 Анализ популярных СУБД. URL: https://drach.pro/blog/hi-tech/item/196-popular-relational-dbms-2022 (дата обращения: 28.04.2022).

6 Белоусова С.А., Рогозов Ю.И. Анализ подходов к созданию пользовательского интерфейса // Известия ЮФУ. Технические науки. 2014. №6 (155). URL: https://cyberleninka.ru/article/n/analiz-podhodov-k-sozdaniyu-polzovatelskogo-interfeysa (дата обращения: 22.04.2022).

7 Документация по Axios. URL: https://axios-http.com/docs/intro (дата обращения: 11.06.2022).

8 Документация по Babel. URL: https://babeljs.io/ (дата обращения: 23.04.2022).

9 Документация по Flow. URL: https://flow.org/ (дата обращения: 22.04.2022).

10 Документация по JavaScript. URL: https://developer.mozilla.org/ru/docs/Web/JavaScript (дата обращения: 22.04.2022).

11 Документация React. URL: https://ru.reactjs.org/docs/introducing-jsx.html (дата обращения: 18.05.2022).

12 Документация по React Native. URL: https://reactnative.dev/ (дата обращения 06.06.2022).

13 Документация по Realm. URL: https://www.mongodb.com/docs/realm/sdk/react-native/ (дата обращения: 28.05.2022).

14 Документо-ориентированная СУБД. URL: https://ru.wikipedia.org/wiki/Документоориентированная_СУБД (дата обращения: 26.05.2022).

15 Кривая забывания Эббингауза. URL: https://ru.wikipedia.org/wiki/Кривая_забывания (дата обращения: 24.05.2022).

16 Минимальный жизнеспособный продукт. URL: https://habr.com/ru/company/productstar/blog/508892/ (дата обращения: 03.06.2022).

17 Официальный сайт ФГБОУ ВО «ХГУЭП». URL: http://www.ael.ru/ (дата обращения: 20.04.2022)

18 Официальный сайт Figma. URL: https://www.figma.com/ (дата обращения: 28.04.2022).

19 Официальный сайт Reword. URL: https://reword.app/ru/en (дата обращения: 15.05.2022).

20 Проблемы изучения английского языка. URL: https://clp.ru/top-9-problem.html (дата обращения: 12.05.2022).

21 Размер словарного запаса. URL: https://skyeng.ru/articles/razmer-slovarnogo-zapasa-skolko-anglijskih-slov-nuzhno-znat-dlya-svobodnogo-vladeniya-yazykom/ (дата обращения: 26.05.2022).

22 Расчёт трудоёмкости работ. URL: https://assistentus.ru/vedenie-biznesa/trudoemkost-rabot/ (дата обращения: 22.05.2022).

23 Статистика рынка мобильных приложений. URL: https://asomobile.net/blog/digest2020/ (дата обращения 17.05.2022).

24 Экономическая эффективность. URL: https://ru.wikipedia.org/wiki/Экономическая_эффективность (дата обращения: 02.06.2022).

25 Android Studio. URL: https://ru.wikipedia.org/wiki/Android_Studio (дата обращения: 26.05.2022).

26 API Яндекс словаря. URL: https://yandex.ru/dev/dictionary/ (дата обращения: 22.05.2022).

27 Babel. URL: https://en.wikipedia.org/wiki/Babel_(transcompiler) (дата обращения: 17.05.2022).

28 Figma – простое решение для дизайнера, сложное решение для верстальщика. URL: https://habr.com/ru/post/463181/ (дата обращения: 07.05.2022).

29 Figma. URL: https://ru.wikipedia.org/wiki/Figma (дата обращения: 05.06.2022).

30 Flutter. URL: https://ru.wikipedia.org/wiki/Flutter (дата обращения: 17.05.2022).

31 Free Dictionary API. URL: https://dictionaryapi.dev/ (дата обращения: 11.06.2022).

32 Oxford Dictionaries API. URL: https://developer.oxforddictionaries.com/documentation (дата обращения: 22.05.2022).

33 React Native. URL: https://ru.wikipedia.org/wiki/React_Native (дата обращения: 17.05.2022).

34 React Native Async storage. https://react-native-async-storage.github.io/async-storage/ (дата обращения: 15.05.2022).

35 React Native Bridge. URL: https://www.geeksforgeeks.org/what-is-a-bridge-in-react-native/ (дата обращения: 20.05.2022).

36 React Native Vector Icons. URL: https://github.com/oblador/react-native-vector-icons (дата обращения: 28.05.2022).

37 Realm. URL: https://en.wikipedia.org/wiki/Realm_(database) (дата обращения: 01.06.2022).

38 TypeScript. URL: https://ru.wikipedia.org/wiki/TypeScript (дата обращения: 08.06.2022).

39 Visual Studio Code. URL: https://ru.wikipedia.org/wiki/Visual_Studio_Code (дата обращения: 25.05.2022).

40 Yarn URL: https://en.wikipedia.org/wiki/Yarn_(package_manager) (дата обращения 25.05.2022).

ПРИЛОЖЕНИЕ А

Исходный код компонента App.js

```
/**
 * @flow strict
 *
 * Created by Nikita Fetisoff (OriginatorX)
 */
import SplashScreen from 'react-native-splash-screen';
import type { Node } from 'react';
import React from 'react';
import { StatusBar} from 'react-native';
import changeNavigationBarColor from 'react-native-navigation-bar-color';
import RootNavigation from './navigation';
import { appThemes } from './styles';
import ThemeContext from './context';
import type { ColorSchema } from './types';
import { SafeAreaProvider } from 'react-native-safe-area-context';
import AsyncStorage from '@react-native-async-storage/async-storage';
type State = {
    theme: ColorSchema,
    toggleTheme: () => void
};
export default class App extends React.Component<void, State> {
  state: State = {
        theme: appThemes.light,
        toggleTheme: this.handleToggleTheme()
    }
    componentDidMount() {
        this.getStoredAppTheme();
        this.customizeApp();
        SplashScreen.hide();
    }
    componentDidUpdate() {
        this.customizeApp();
        this.storeAppTheme(this.state.theme);
    }
    storeAppTheme(value: ColorSchema): void {
        const jsonValue = JSON.stringify(value);
        AsyncStorage.setItem('@app_theme_value', jsonValue)
            .then()
            .catch((e: Error) => {
                console.log(e.message);
            });
    }
    getStoredAppTheme(): void {
        AsyncStorage.getItem('@app_theme_value')
            .then(json => {
                if (json !== null) {
                    const theme = JSON.parse(json);
                    if (theme !== undefined) {
                        this.setState({theme});
                    }
                    return;
                }
            })
```

```
            .catch((e: Error) => {
                console.log(e.message);
            });
    }
    customizeApp() {
        const theme = this.state.theme;
        StatusBar.setBackgroundColor(theme.mainElementColor, true);
        StatusBar.setBarStyle(theme.statusBarContent, true);
        changeNavBarTheme(theme.mainElementColor, theme.navBarLignt);
    }
    handleToggleTheme(): Function {
        return function toggleTheme() {
            this.setState(preventState => ({
                theme:
                    preventState.theme === appThemes.light
                        ? appThemes.dark
                        : appThemes.light,
            }));
        }.bind(this);
    }
    render(): Node {
        return (
            <SafeAreaProvider>
                <ThemeContext.Provider
                    value={{
                        theme: this.state.theme,
                        toggleTheme: this.state.toggleTheme
                    }}
                >
                    <RootNavigation />
                </ThemeContext.Provider>
            </SafeAreaProvider>
        );
    }
}
async function changeNavBarTheme(
    color: string,
    light: boolean = true,
    animated: boolean = true
) {
    try {
        await changeNavigationBarColor(color, light, animated);
    } catch(e) {
        console.log(e.message);
    }
}
```

ПРИЛОЖЕНИЕ Б

Исходный код экрана настроек

```
/**
 * @flow strict
 */
import type { Node } from 'react';

import React, { Component } from 'react';
import {
   View,
   Text,
   TouchableOpacity,
   Switch
} from 'react-native';
import { NavHeader } from '../../components';
import styles from './Settings.style';
import { activeOpacity } from '../../constants';
import Icon from 'react-native-vector-icons/MaterialCommunityIcons';
import ThemeContext from '../../context';
import type { ColorSchema } from '../../types';

type SettingsProps = {
   navigation?: any
};

type State = {
   btnEnabled: boolean
};

export default class Settings extends Component<SettingsProps, State> {

   static contextType: typeof ThemeContext = ThemeContext;

   handleToggleTheme: Function = function(): void {
      this.context.toggleTheme();
   };

   render(): Node {

      const theme: ColorSchema = this.context.theme;
      const enable: boolean = this.context.theme.switchThemeActive;
      return(
         <View style={[styles.container, {backgroundColor: theme.background}]}>
            <NavHeader
               isBackArrow
               text="Настройки"
               textStyle={{color: theme.text}}
               style={{backgroundColor: theme.mainElementColor, elevation: 10}}
               navigation={this.props.navigation}
            />
            <View style={styles.settings}>
               <ItemOfSettings label="Темная тема">
                  <Switch
                     style={styles.switchScale}
                     trackColor={{ false: "#767577", true: "#767577" }}
                     thumbColor={enable ? "#FFB310" : "#212C34"}
```

```
                ios_backgroundColor="#3e3e3e"
                onValueChange={this.handleToggleTheme.bind(this)}
                value={enable}/>
            </ItemOfSettings>
            <View style={styles.colorModes}>
              <Text style={[
                styles.colorModesTitle,
                {color: theme.text}
              ]}>
                Цветовые моды
              </Text>
              <View style={styles.modes}>
                <ModeContainer label="Yellow" color="#FFB310" />
                <ModeContainer label="Blue" color="#293462" />
                <ModeContainer label="Dark" color="#212C34" />
                <ModeContainer label="Purple" color="#A94CAF" />
              </View>
            </View>
            <ItemOfSettings
              label="Информация о приложении"
              touchable
            >
              <Icon name="help-circle-outline" size={25} color={theme.text} />
            </ItemOfSettings>
            <ItemOfSettings
              label="Настройки уведомлений"
              touchable
            >
              <Icon name="bell-ring-outline" size={25} color={theme.text} />
            </ItemOfSettings>
            <ItemOfSettings
              label="Изменить язык"
              touchable
            >
              <Icon name="translate" size={25} color={theme.text} />
            </ItemOfSettings>
          </View>
        </View>
      );
    }
}

type ModeContainerProps = {
  label: string,
  color: string
};

function ModeContainer({label, color}: ModeContainerProps): Node {
  return (
    <ThemeContext.Consumer>
      {({theme}: {theme: ColorSchema}) => (
        <View style={styles.modeContainer}>
          <TouchableOpacity
            style={[
              styles.modeTouch,
              {backgroundColor: color}
            ]}
            activeOpacity={activeOpacity}
          />
```

```
                <Text style={[
                    styles.settingsModesLabels,
                    {color: theme.text}
                ]}>
                    {label}
                </Text>
            </View>
        )}
    </ThemeContext.Consumer>
  )
}

// This must be outside!!!

type ItemOfSettingsProps = {
  label: string,
  children?: Node,
  touchable?: boolean
};

function ItemOfSettings({
  label, children, touchable = false
}: ItemOfSettingsProps): Node {

  const Content = ({theme}: {theme: ColorSchema}): Node => (
    <>
      <View style={styles.settingsThemeTitle}>
        <Text style={[
          styles.settingsThemeTxt,
          {color: theme.text}
        ]}>
          {label}
        </Text>
      </View>
      {children}
    </>
  );

  return (
    <ThemeContext.Consumer>
      {({theme}: {theme: ColorSchema}) => (
        touchable
          ? (
            <TouchableOpacity
              style={styles.settingsAppTheme}
              activeOpacity={activeOpacity}
            >
              <Content theme={theme} />
            </TouchableOpacity>
          )
          : (
            <View style={styles.settingsAppTheme}>
              <Content theme={theme} />
            </View>
          )
      )}
    </ThemeContext.Consumer>
  );
}
```

ПРИЛОЖЕНИЕ В

Исходный код навигационного бара

```
/**
 * @flow strict
 */
import type { Node } from 'react';
import React, { Component } from 'react';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import Home from '../screens/Home/Home';
import Dictionaries from '../screens/Dictionaries/Dictionaries';
import Settings from '../screens/Settings/Settings';
import Words from '../screens/Words/Words';
import Search from '../screens/Search/Search';
import AddingAWord from '../screens/AddingAWord/AddingAWord';
import type {
   MaterialCommunityIconsGlyphs
} from 'react-native-vector-icons/MaterialCommunityIcons';
import ThemeContext from '../context';
import type { ColorSchema, ThemeContextType } from '../types';
import Icon from 'react-native-vector-icons/MaterialCommunityIcons';
import { NavigationContainer } from '@react-navigation/native';

type MenuIconProps = {
   color: string
};

function setIcon(route: any, size: number): (props: MenuIconProps) => Node {
   return function MenuIcon({color}: MenuIconProps): Node {
      let iconName: MaterialCommunityIconsGlyphs = 'home-outline';
      switch(route.name) {
         case 'Search':
            iconName = 'magnify'
         break;
         case 'Dict':
            iconName = 'bookshelf'
         break;
      }
      return <Icon name={iconName} size={size} color={color} />;
   }
}

const BottomTab = createBottomTabNavigator();

class BottomNavBar extends Component<{navigation?: any}> {

   static contextType: ThemeContextType = ThemeContext;

   render(): Node {
      const theme: ColorSchema = this.context.theme;
      return (
         <BottomTab.Navigator
            initialRouteName="Home"
            screenOptions={
               ({route}) => ({
                  tabBarIcon: setIcon(route, 30),
```

```
                headerShown: false,
                tabBarShowLabel: false,
                tabBarStyle: {
                  backgroundColor: theme.mainElementColor,
                  borderTopWidth: 0,
                  elevation: 10,
                },
                tabBarActiveTintColor: theme.text,
                tabBarInactiveTintColor: '#747474',
              })
            }
          >
            <BottomTab.Screen
              name="Home"
              component={Home}
            />
            <BottomTab.Screen
              name="Search"
              component={Search}
            />
            <BottomTab.Screen
              name="Dict"
              component={DictScreen}
            />
          </BottomTab.Navigator>
        );
    }
}
const DictStack = createNativeStackNavigator();

function DictScreen(): Node {
    return (

        <DictStack.Navigator
          screenOptions={{headerShown: false}}
        >
          <DictStack.Screen
            name="Dictionaries"
            component={Dictionaries}
          />
          <DictStack.Screen
            name="Words"
            component={Words}
          />
        </DictStack.Navigator>
    );
}
const MainStack = createNativeStackNavigator();
export default class Navigation extends Component<{navigation?: any}> {
    render(): Node {
        return (
          <NavigationContainer>
            <MainStack.Navigator
              initialRouteName="Root"
              screenOptions={{
                headerShown: false
              }}
            >
              <MainStack.Screen
```

```
                name="Root"
                component={BottomNavBar}
              />
              <MainStack.Screen
                name="Settings"
                component={Settings}
              />

              <MainStack.Group screenOptions={{
                presentation: 'modal',
              }}>
                <MainStack.Screen
                  name="AddingAWord"
                  component={AddingAWord}
                />
              </MainStack.Group>
            </MainStack.Navigator>
          </NavigationContainer>
      )
    }
}
```

ПРИЛОЖЕНИЕ Г

Исходный код компонента Dictionaries

```
/**
 * @flow strict
 */
import type { Node } from 'react';

import React, {Component} from 'react';
import { View } from 'react-native';
import {
    ItemOfList,
    CustomBtn,
    NavHeader
} from '../../components';
import styles from './Dictionaries.style';
import { blackPlus } from '../../assets/icons';
import ThemeContext from '../../context';
import type { ColorSchema, ThemeContextType } from '../../types';
import realm, { Dict } from '../../db';
import type { DictType, WordType } from '../../types';
import { SwipeList, ModalWrapper, TouchableIcon } from '../../components';
import CollectionSearch from '../../components/CollectionSearch';
import ModalWindow from './ModalWindow';
import { Results } from 'realm';
type State = {
    modalIsOpen: boolean,
    dicts: Results<DictType>,
    words: Results<WordType>,
    searchRequest: string
};
export default class Dictionaries extends Component<any, State> {
    static contextType: ThemeContextType = ThemeContext;
    state: State = {
        modalIsOpen: false,
        dicts: realm.objects('Dict').sorted('createdAt', true),
        words: realm.objects('Word').sorted('createdAt', true),
        searchRequest: ''
    };
    handleUpdateDicts(): void  {
        this.setState({
            dicts: realm.objects('Dict').sorted('createdAt', true)
        });
    }
    handleModalVisibility: Function = function(): void {
        this.setState(preventState => ({
            modalIsOpen: !preventState.modalIsOpen
        }));
    };
    handleSearchRequest: Function = function(request: string): void {
        this.setState({
            searchRequest: request
        });
    };
    handleAddDict: Function = function(name: string): void {
        realm.write(() => {
            realm.create('Dict', new Dict({
```

```
            name,
            wordsAmount: this.state.words.length,
            icon: 'orangeDictionary',
            words: this.state.words
        }));
    });
};
handleDeleteDict: Function = function(dict: DictType): void {
    realm.write(() => {
        realm.delete(dict);
    });
    this.handleUpdateDicts();
};
render(): Node {
    const theme: ColorSchema = this.context.theme;
    let visibleDicts: Array<DictType> = [];
    this.state.dicts.forEach(dict => {
        let name: string = dict.name.toLowerCase();
        let request: string = this.state.searchRequest.toLowerCase();

        if (name.indexOf(request) !== -1) {
            visibleDicts.push(dict);
        }
    });
    return (
        <View style={[styles.dict, {backgroundColor: theme.background}]}>
            <NavHeader
                text="Словари"
                textStyle={[styles.dictTitle, {color: theme.text}]}
                style={[styles.dictHeaderStyle, {backgroundColor: theme.mainElementColor}]}
            >
                <TouchableIcon icon="format-list-text" iconSize={25} />
            </NavHeader>
            <SwipeList
                ListHeaderComponent={
                    <CollectionSearch
                        value={this.state.searchRequest}
                        searchRequest={this.state.searchRequest}
                        handleSearchRequest={this.handleSearchRequest.bind(this)}
                    />
                }
                data={visibleDicts}
                renderItem={
                    ({item}: {item: DictType}): Node => (
                        <ItemOfList
                            styles={styles}
                            name={item.name}
                            description={String(item.wordsAmount)}
                            leftIcon={item.icon}
                            rightIconName="dots-vertical"
                            dictId={item._id}
                        />
                    )
                }
                onDeleteItem={this.handleDeleteDict.bind(this)}
                theme={theme}
            />
            <CustomBtn
                style={[styles.addBtn, styles.dictAddButtonContainer]}
```

```
            text="Создать"
            image={blackPlus}
            onPress={this.handleModalVisibility.bind(this)}
          />
          <ModalWrapper
            isVisible={this.state.modalIsOpen}
            handleVisibility={this.handleModalVisibility.bind(this)}
          >
            <ModalWindow
              title="Создать словарь"
              handleVisibility={this.handleModalVisibility.bind(this)}
              handleAddDict={this.handleAddDict.bind(this)}
            />
          </ModalWrapper>
        </View>
      );
    }
}
```

ПРИЛОЖЕНИЕ Д

Исходный код компонента AddingAWord

```
/**
 * @flow strict
 */
import type { Node } from 'react';
import React, { Component } from 'react';
import {
   View,
   TextInput,
   ScrollView,
   Image,
   Text,
   TouchableOpacity
} from 'react-native';
import styles from './AddingAWord.style';
import ThemeContext from '../../context';
import type { ThemeContextType, ColorSchema, WordType } from '../../types';
import getWord from '../../middleware/httpRequests';
import { activeOpacity } from '../../constants';
import { CustomBtn, TouchableIcon } from '../../components';
import { branding } from '../../assets/images';
import icons from '../../assets/icons';
import Sound from 'react-native-sound';

const partsOfSpeechToRus: Map<string, string> = new Map([
   ['noun',        'Существительное'],
   ['verb',        'Глагол'],
   ['adverb',      'Наречие'],
   ['adjective',   'Прилагательное'],
   ['pronoun',     'Местоимение'],
   ['numeral',     'Имя числительное'],
   ['preposition', 'Предлог'],
   ['conjunction', 'Союз'],
   ['particle',    'Частица'],
   ['participle',  'Причастие']
]);
const usingLangs = {
   english: 'en-ru',
   russian: 'ru-en'
};
export type AddingAWordProps = {
   addDict?: () => void,
   navigation: any
};
export type State = {
   word: string,
   translate: string,
   transcription: string,
   wordDescription: WordType,
   sound: Sound
}
export default
class AddingAWord extends Component<AddingAWordProps, State> {
   static contextType: ThemeContextType = ThemeContext;
   state: State = {
```

```
    word: '',
    translate: '',
    transcription: '',
    wordDescription: {},
    sound: null
};
componentDidMount() {
    Sound.setCategory('Playback');
}
handleBack: Function = function(): void {
    this.props.navigation.goBack();
};
handleWordInput: Function = function(word: string): void {
    this.setState({word});
};
handleSearchWord: Function = function(): void {
    if (this.state.word !== '') {
        const lang = defineLang(this.state.word);
        getWord(this.state.word, lang)
            .then(word => {
                if (word ?? false) {
                    this.setState({
                        wordDescription: word,
                        sound: this.setSound(
                            word.pronunciation
                        )
                    });
                }
                return;
            })
            .catch((e: Error) => {
                console.log(e.message);
            })
        ;
    }
};
setSound(pronunciation: any): Sound {
    if (pronunciation !== '') {
        return new Sound(
            pronunciation,
            Sound.MAIN_BUNDLE,
            (error: Error) => {
                if (error) {
                    console.log(error.message);
                    return;
                }
            }
        );
    }
}
handlePronunSound: Function = function(): void {
    //this.state.sound.setVolume(1);
    //this.state.sound.play();
};
render(): Node {
    const theme: ColorSchema = this.context.theme;
    return (
        <View
            style={[styles.container, {
```

```
      backgroundColor: theme.background
    }]}
>
    <View
      style={[
        styles.header,
        {
          backgroundColor: theme.mainElementColor,
          height:
            objectIsNotEmpty(this.state.wordDescription)
              ? '95%'
              : 200,
        }
      ]}
    >
      <View style={{height: 200}}>
        <View style={styles.searchHeaderWord}>
          <View style={styles.searchHeaderWordLeft}>
            <TouchableIcon
              icon="arrow-left"
              onPress={this.handleBack.bind(this)}
            />
            <Input
              title="Новое слово..."
              style={styles.searchHeaderWordLeftField}
              handleInput={this.handleWordInput.bind(this)}
            />
          </View>
          <TouchableIcon
            icon="magnify"
            onPress={this.handleSearchWord.bind(this)}
          />
        </View>
        <View style={styles.searchHeaderFields}>
          <Input
            title="Перевод..."
            handleInput={this.handleWordInput.bind(this)}
          />
          <Input
            title="Транскрипция..."
            handleInput={this.handleWordInput.bind(this)}
          />
        </View>
      </View>
      {
        objectIsNotEmpty(this.state.wordDescription)
          ? (
            <>
              <ScrollView
                style={[
                  styles.searchHeaderScrollable,
                  {backgroundColor: theme.mainElementColor,}
                ]}
              >
                {
                  fillPosContainer(
                    this.state.wordDescription,
                    this.state.sound
                  )
```

```
                        }
                      </ScrollView>
                      <View style={styles.addBtnContainer}>
                        <CustomBtn
                          style={styles.addBtn}
                          image={icons['blackPlus']}
                        />
                      </View>
                    </>
                  ) : null
                }
              </View>
              {
                !(objectIsNotEmpty(this.state.wordDescription))
                  ? (
                    <View style={styles.imageFiller}>
                      <Image
                        source={branding}
                        style={styles.imageFillerSize}
                      />
                    </View>
                  )
                  : null
              }
            </View>
        );
    }
}
function fillPosContainer(wordDescription: WordType, audio: Sound): Array<Node> {
    const PosBlock = (
        {title, words}: {title: string, words: Array<string>}
    ): Node => (
        <ThemeContext.Consumer>
            {({theme}: {theme: ColorSchema}) => (
                words.length !== 0
                    ? (
                        <>
                            <Text
                                style={[
                                    styles.title,
                                    {color: theme.text}
                                ]}
                            >
                                {title}:
                            </Text>
                            <View style={styles.posContainer}>
                                {fillInWithWords(words)}
                            </View>
                        </>
                    )
                    : null
            )}
        </ThemeContext.Consumer>
    );
    const playAudio = (): void => {
        audio.setVolume(1);
        audio.play();
    };
    const Pronunciation = (): Node => (
```

```
        <ThemeContext.Consumer>
          {({theme}: {theme: ColorSchema}) => (
            <View style={styles.posContainerPronun}>
              <Text style={[styles.title, {color: theme.text}]}>
                Произношение:
              </Text>
              <TouchableIcon
                key={"volume-high"}
                icon="volume-high"
                iconSize={30}
                onPress={playAudio}
              />
            </View>
          )}
        </ThemeContext.Consumer>
      );
      let blocksWithWords: Array<Node> = [];
      if (audio ?? false) {
        blocksWithWords.push(
          <Pronunciation key={'pronunciation_key'}/>
        );
      }
      for (let [pos, words] of wordDescription?.partsOfSpeech) {
        blocksWithWords.push(
          <PosBlock
            key={pos}
            title={String(partsOfSpeechToRus.get(pos))}
            words={words}
          />
        );
      }
      return blocksWithWords;
    }
    function fillInWithWords(words: Array<string>): Array<Node> {
      const TouchableWord = ({text}: {text: string}): Node => (
        <ThemeContext.Consumer>
          {({theme}: {theme: ColorSchema}) => (
            <TouchableOpacity
              activeOpacity={activeOpacity}
              style={styles.posContainerWords}
            >
              <Text
                style={[
                  styles.posContainerTxt,
                  {color: theme.text}
                ]}
              >
                {text}
              </Text>
            </TouchableOpacity>
          )}
        </ThemeContext.Consumer>
      );
      const wordsList: Array<Node> = [];
      for (let word of words) {
        wordsList.push(
          <TouchableWord key={word} text={word} />
        );
      }
```

```
    return wordsList;
}
type InputProps = {
    title: string,
    style?: Object,
    handleInput: (word: string) => void
};
function Input({
    title, style, handleInput,
}: InputProps): Node {
    return (
        <ThemeContext.Consumer>
            {({theme} : {theme: ColorSchema}) => (
                <TextInput
                    style={[
                        styles.searchHeaderInput,
                        {
                            color: theme.text,
                            ...style
                        }
                    ]}
                    placeholder={title}
                    placeholderTextColor="#747474"
                    onChangeText={handleInput}
                />
            )}
        </ThemeContext.Consumer>
    );
}
function objectIsNotEmpty(obj: {}): boolean {
    if (typeof obj !== 'function') {
        return Object.keys(obj).length !== 0
    }
    return false;
}
function defineLang(text: string): string {
    let asciiCode: number = text.charCodeAt(0);
    return (
        asciiCode >= 'A'.charCodeAt(0)
        && asciiCode <= 'я'.charCodeAt(0)
            ? usingLangs.russian
            : usingLangs.english
    );
}
```