Министерство науки и высшего образования РФ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Уральский государственный педагогический университет»
Институт математики, физики, информатики и технологий
Кафедра информационно-коммуникационных технологий в образовании

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ НА ANDROID ДЛЯ ГЕНЕРАЦИИ SQL КОДА

Выпускная квалификационная работа
бакалавра по направлению подготовки
09.03.02 – Информационные системы и технологии

Исполнитель: студент группы ИСиТ-1501
Института математики, физики, информатики
и технологий
Иванов А.А.

Руководитель: к.п.н., доцент кафедры ИКТО
Сардак Л.В.

Работа допущена к защите
«20» мая 2019 г.
Зав. кафедрой _____

Екатеринбург – 2019

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время существует большое количество сетевых информационных систем, позволяющих моделировать базы данных и генерировать SQL код для их развертывания в системах управления базами данных. Самые популярные из них «DB Designer», «SQL Database Modeler», «dbdiagram.io» и прочие. Воспользоваться данными информационными системами можно только с помощью браузера, они не имеют аналогов в формате мобильного приложения. Такой способ взаимодействия не всегда удобен так как, например, браузеры для мобильных устройств неполноценно поддерживают технологию «drag and drop». Представленные сервисы не адаптированы к их использованию с помощью мобильных браузеров с их ограниченным функционалом. На данном этапе развития информационных технологий, взаимодействие человека и информационных систем, посредством мобильных устройств, в основном осуществляется с помощью мобильных приложений.

Таким образом, представляется актуальным провести разработку приложения, реализующего технологию проектирования базы данных с помощью мобильных устройств.

Предмет разработки – мобильное приложение для проектирования баз данных и генерации SQL кода «EasyBox», для устройств, работающих под управлением операционной системы Android версии 7.1 и выше.

Цель работы – разработать и описать мобильное приложение для проектирования и генерации SQL кода «EasyBox» под управлением операционной системы Android и описать его.

Задачи:

1. Произвести анализ существующих технологий, позволяющих на основе спроектированной базы данных генерировать SQL код, а также сред разработок и языков программирования. Обосновать выбор технологии реализации приложения.
2. Сформулировать техническое задание на разработку приложения.

3. В соответствии с техническим заданием построить информационные, функциональные диаграммы и их декомпозиции, отражающие взаимодействие и функционал компонентов системы.

4. Провести апробацию мобильного приложения.

ГЛАВА 1. АНАЛИЗ ТЕХНОЛОГИЙ РАЗРАБОТКИ ПРИЛОЖЕНИЙ ДЛЯ ОС ANDROID ПО РАБОТЕ С SQL КОДОМ

1.1 Технологии проектирования баз данных

Проектирование баз данных – это процесс создания схемы (модели) базы данных и определения необходимых ограничений целостности. Проектирование базы данных является фундаментальным этапом, который лежит в основе разработки любого прикладного программного обеспечения. Проектирование БД обеспечивает такие задачи как: целостность данных, хранение информации, доступ к данным, сокращение избыточности и дублирования данных.

Проектирование баз данных делиться на этапы: концептуальное, логическое и физическое. Рассмотрим каждый из этапов подробнее.

- концептуальное проектирование представляет собой построение семантической модели какой-либо из предметных областей. Такое проектирование является наивысшим уровнем абстракции, на данном этапе не учитывается целевая система управления базами данных. Обычно для разработки концептуальной модели базы данных используются ER-диаграммы. Концептуальная модель содержит в себе описание основных объектов какой-либо предметной области и их взаимодействие друг с другом;
- логическое проектирование представляет собой построение схемы базы данных на основе какой-то определенной модели данных. Примером такой модели данных может послужить реляционная модель данных, для этой модели логическая модель представляет собой набор различных схем-отношений с указанием первичных ключей и связей между отношениями в виде внешних ключей. Переход от концептуальной модели к логической происходит по формальным правилам. Данный этап может быть в большей степени автоматизирован. На этом этапе проектирования учитывается специфика определенной модели данных, но, как и на концептуальном

этапе может не учитываться особенности определённой системы управления базами данных;

- физическое проектирование — это создание схемы базы данных для определенной системы управления базами данных. Особенности и свойства определенной СУБД может содержать определенные ограничение, такие как: наименование объектов, поддерживаемые типы данных и т. п. Кроме того, особенности определенной СУБД при физическом проектировании включает в себя определение таких вещей как: выбор методов управления памятью на дисках, разделение базы данных по устройствам и файлом, методов доступа к данным, создание индексов и т. д. Результатом этого этапа является выходной SQL код под конкретную СУБД [44].

В настоящее время существует большое количество онлайн сервисов позволяющих спроектировать базу данных с учётом всех трёх этапов: от построения семантической модели, до генерации SQL кода под требуемую СУБД. Проведем анализ представленных технологий.

Онлайн сервис DbDesigner [15] разработан одноименной компанией DB Designer, которая была основана в 2006 году разнообразной командой инженеров и разработчиков. DbDesigner используют более 40 000 организаций, от ведущих государственных учреждений до фирм корпоративного класса, небольших компаний и более 100 000 фрилансеров и разработчиков. У этой компании существует, так называемая, академическая инициатива. Она обеспечивает бесплатный доступ для преподавателей и студентов по всему миру. DB Designer является академическим партнером 278 университетов и колледжей по всему миру, в которых более 2000 профессоров с 2006 года обучили более 50 000 студентов по предметам, связанным с базами данных, с помощью онлайн сервиса DbDesigner. Также данный сервис позволяет проектировать базы данных без знаний языка

SQL. В данный момент DbDesigner поддерживает генерацию SQL кода для 5 СУБД:

- PostgreSQL;
- SQLite;
- MySQL;
- Microsoft SQL Server;
- Oracle.

Данный онлайн сервис имеет минималистичный и понятный интерфейс (Рис. 1). Также сервис является условно бесплатным: при наличии базового аккаунта можно создать до двух моделей баз данных, включающих в каждую до 10 таблиц. Основная локализация на английском языке, без возможности смены. Вывод данных возможен в трёх форматах: SQL код (Рис. 2), изображения и PDF (Рис. 3).

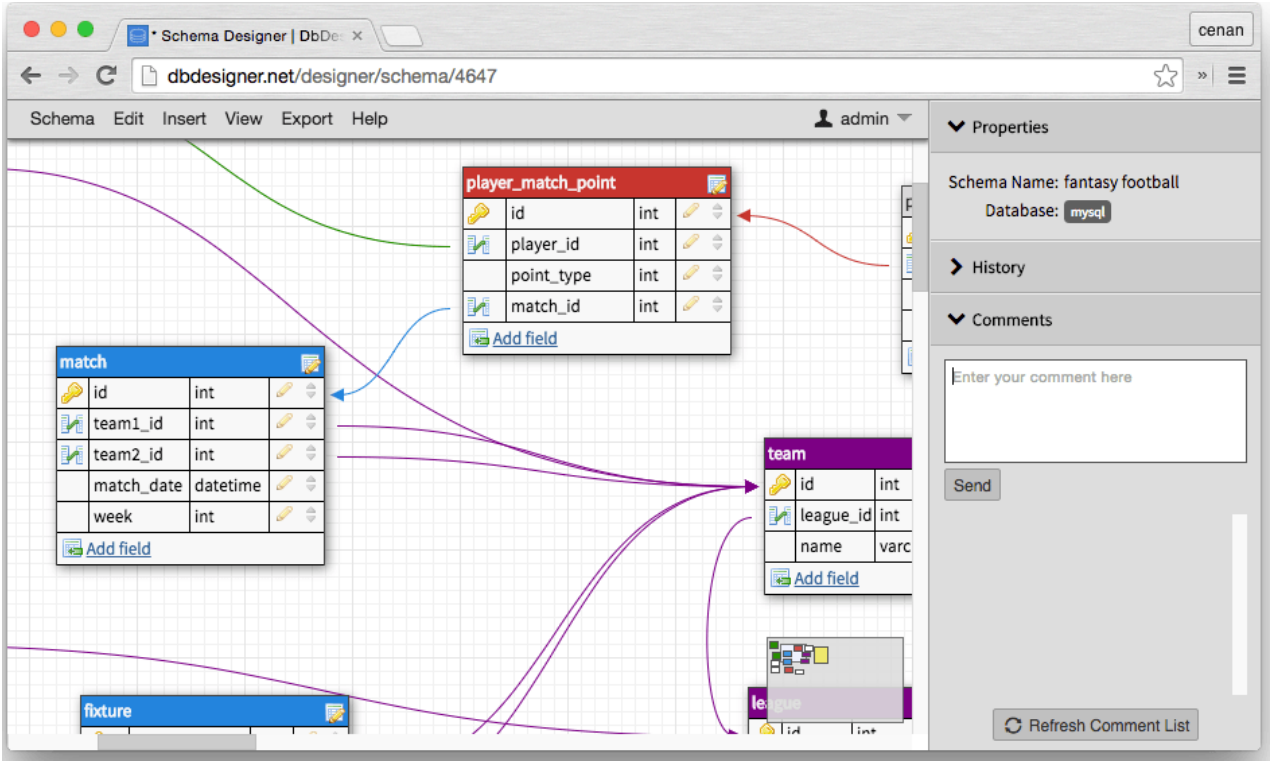

Рис. 1. Интерфейс DbDesigner

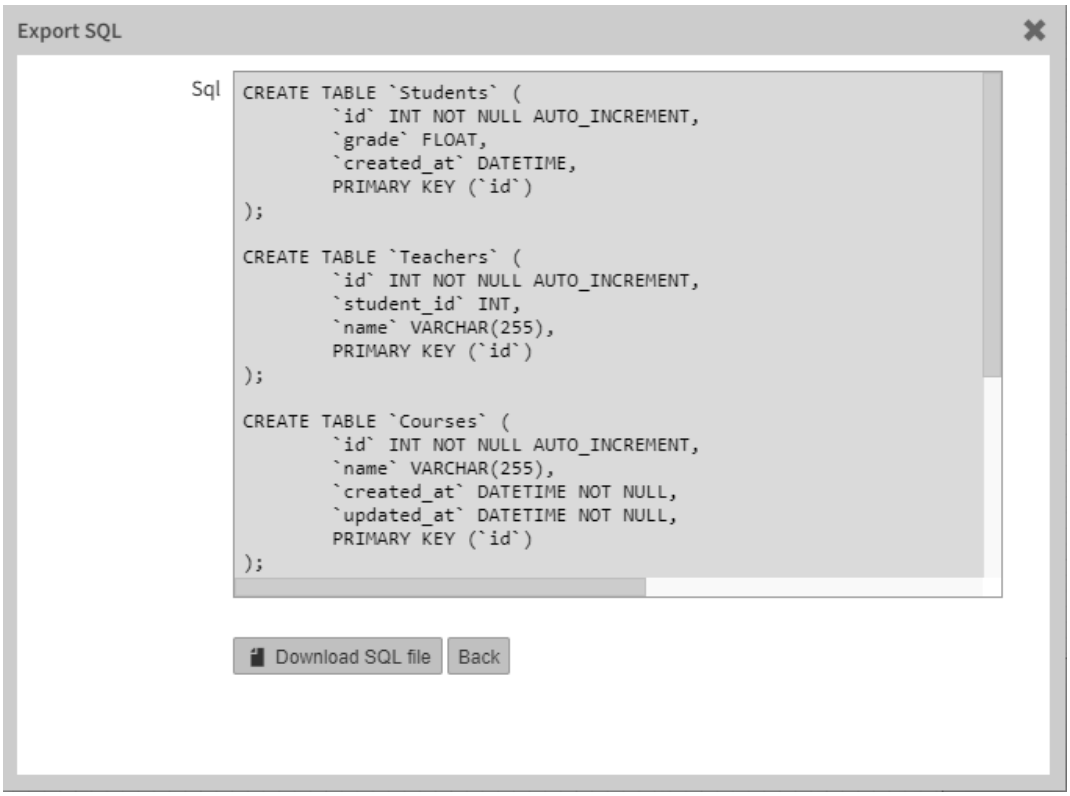

Рис. 2. Генерация SQL кода

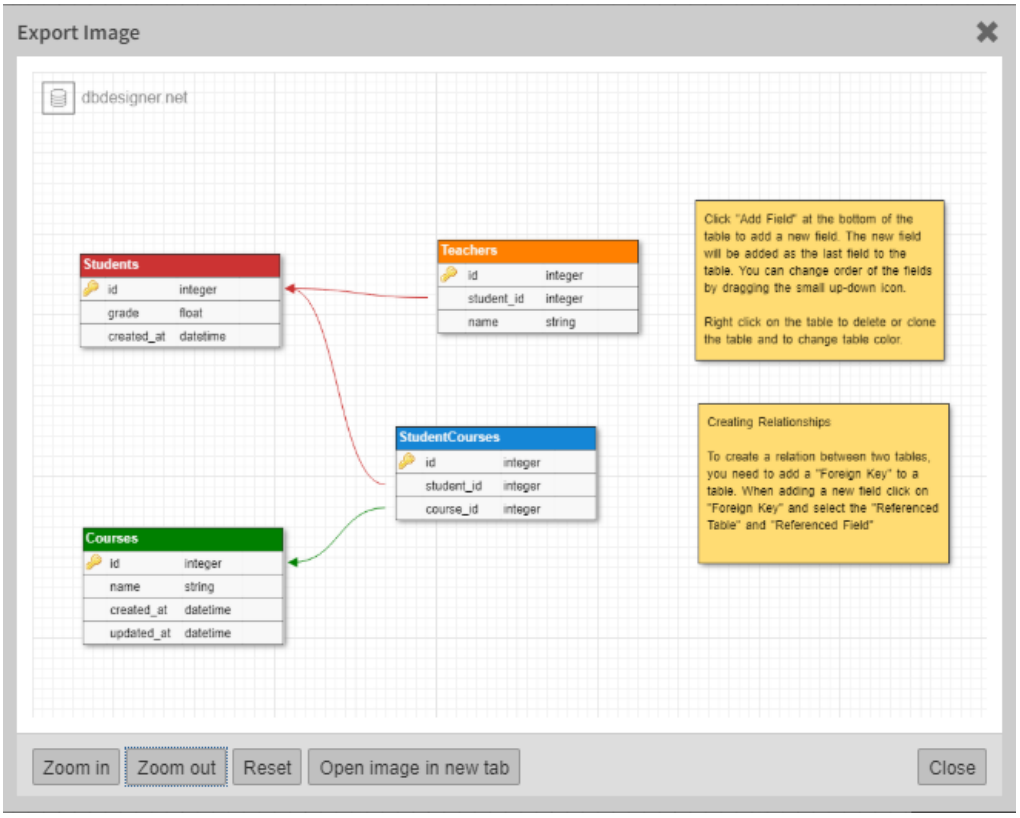

Рис. 3. Сохранение в формате изображения.

У DbDesigner отсутствует аналог в виде мобильного приложения, а версия сайта с мобильного браузера не адаптирована под мобильные устройства (Рис.

4), что делает проектирование баз данных с помощью этого сервиса посредством мобильных устройств затруднительным.

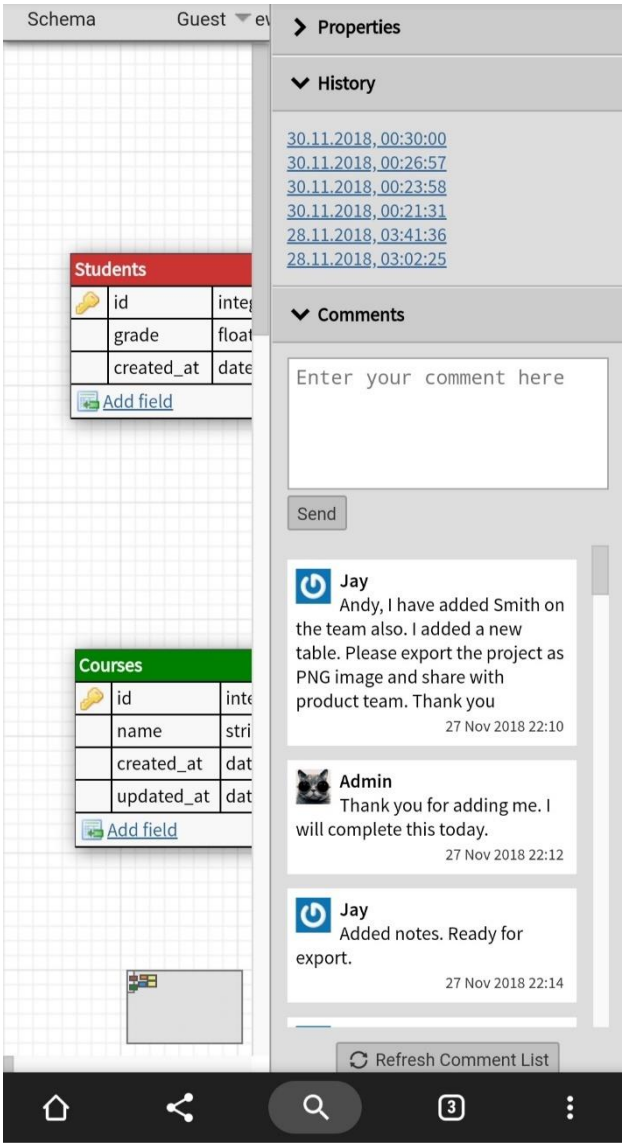

Рис. 4. Мобильная версия сайта.

SQL Database Modeler (SqlDBM) [30] – это условно бесплатный онлайн сервис который позволяет проектировать базы данных без знания языка SQL. Данный сервис используют более 12 000 компаний в более чем 200 странах. Бесплатный (базовый) аккаунт позволяет иметь 1 активный проект и всего 2 таблицы для генерации SQL кода. В SqlDBM доступна генерация SQL кода для трёх различных СУБД:

- Microsoft SQL Server;
- MySQL;

- Snowflake.

SqlDBM имеет широкий функционал и достаточно запутанный интерфейс с множеством вкладок (Рис. 5).

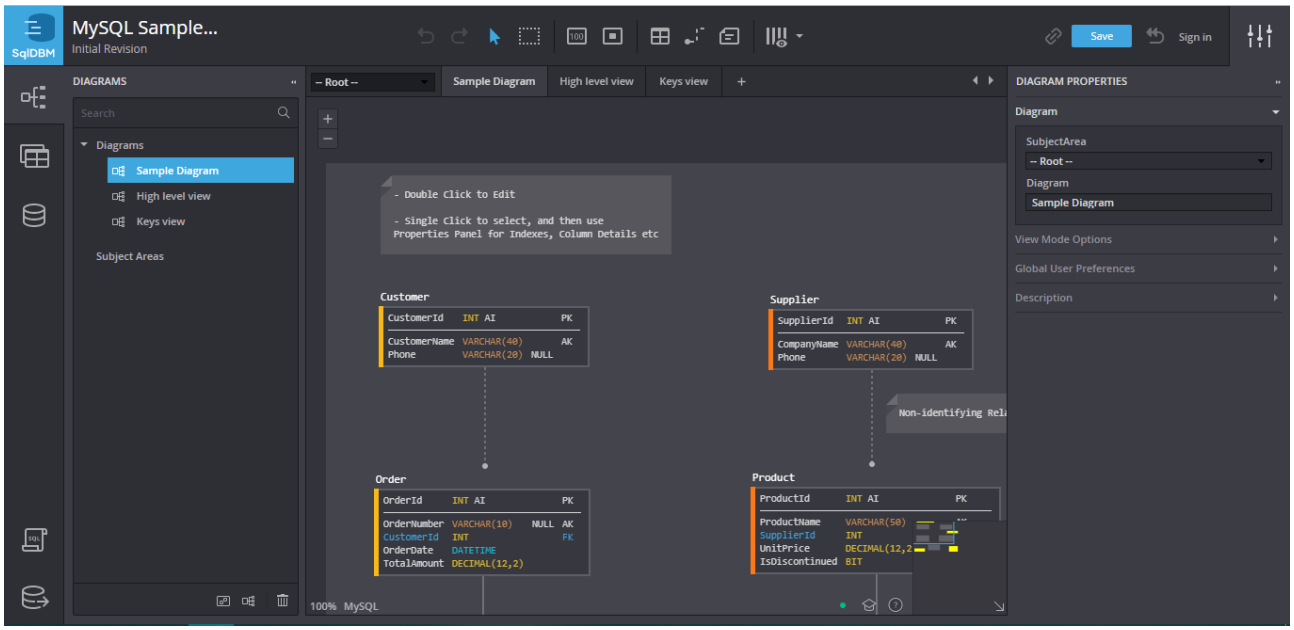

Рис. 5. Интерфейс SqlDBM

Данный онлайн сервис поддерживает вывод данных только в формате SQL кода (Рис. 6).

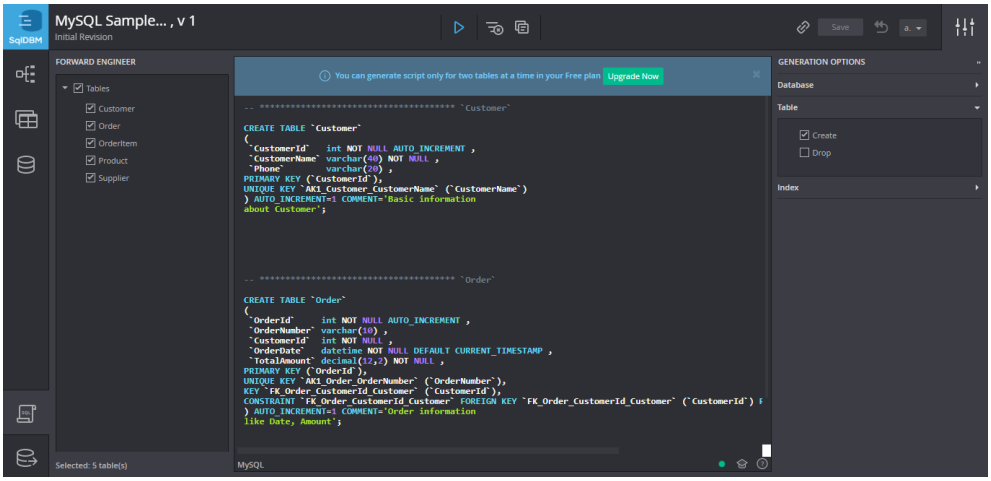

Рис. 6. SqlDBM генерация кода

SqlDBM также не имеет своего мобильного приложения для операционной системы Android, а версия сайта с мобильного браузера не адаптирована под мобильные устройства (Рис. 7).

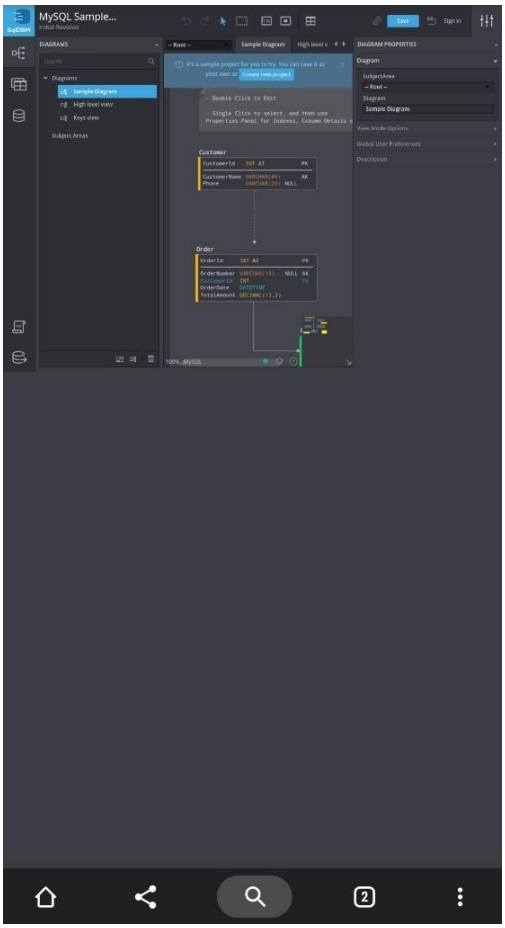

Рис. 7. Интерфейс SqlDBM с мобильного браузера Chrome

Бесплатный онлайн сервис dbdiagram.io [21] также предназначен для полноценного проектирования базы данных с учётом всех этапов проектирования. В отличие от ранее рассмотренных сервисов, чтобы воспользоваться dbdiagram.io необходимо базовые знания языка SQL, так как сущности базы данных (таблицы, поля, связи) задаются не в интерфейсе, а в виде SQL кода. Также данный сервис имеет отличительную черту он полностью бесплатный и не накладывает на пользователей никаких дополнительных ограничений. Имеет интуитивно понятный интерфейс (Рис. 8), но, как и в предыдущих сервисах отсутствуют какие-либо локализации кроме английской.

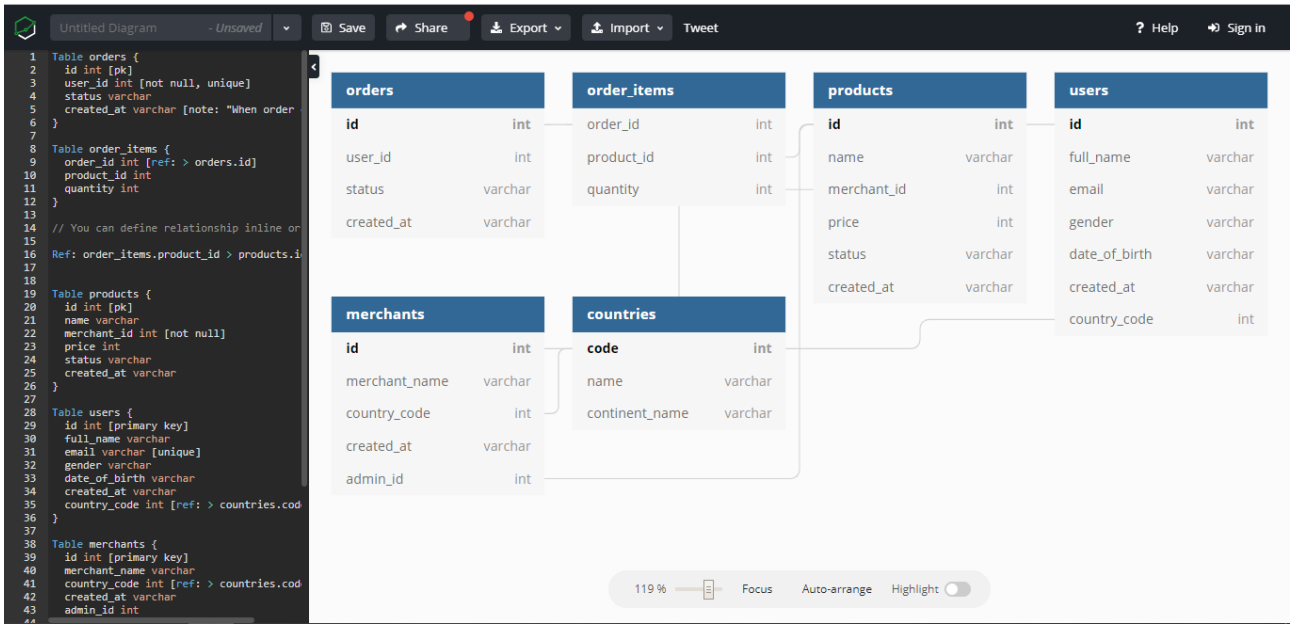

Рис. 8. Интерфейс dbdiagram.io

Сервис dbdiagram.io поддерживает генерацию SQL кода для MySQL и PostgreSQL

Рис. 9), а также сохранение модели в формате PDF.

```sql
CREATE TABLE `orders`
(
  `id` int PRIMARY KEY,
  `user_id` int UNIQUE NOT NULL,
  `status` varchar(255),
  `created_at` varchar(255) COMMENT 'When order created'
);

CREATE TABLE `order_items`
(
  `order_id` int,
  `product_id` int,
  `quantity` int
);

CREATE TABLE `products`
(
  `id` int PRIMARY KEY,
  `name` varchar(255),
  `merchant_id` int NOT NULL,
  `price` int,
  `status` varchar(255),
  `created_at` varchar(255)
);
```

Рис. 9. Сгенерированный SQL код для MySQL

Мобильная версия сайта dbdiagram.io адаптирована чуть лучше для мобильных устройств, чем у DbDesigner и SqlDBM, но из-за большого количества SQL кода редактировать модели достаточно затруднительно (Рис. 10).

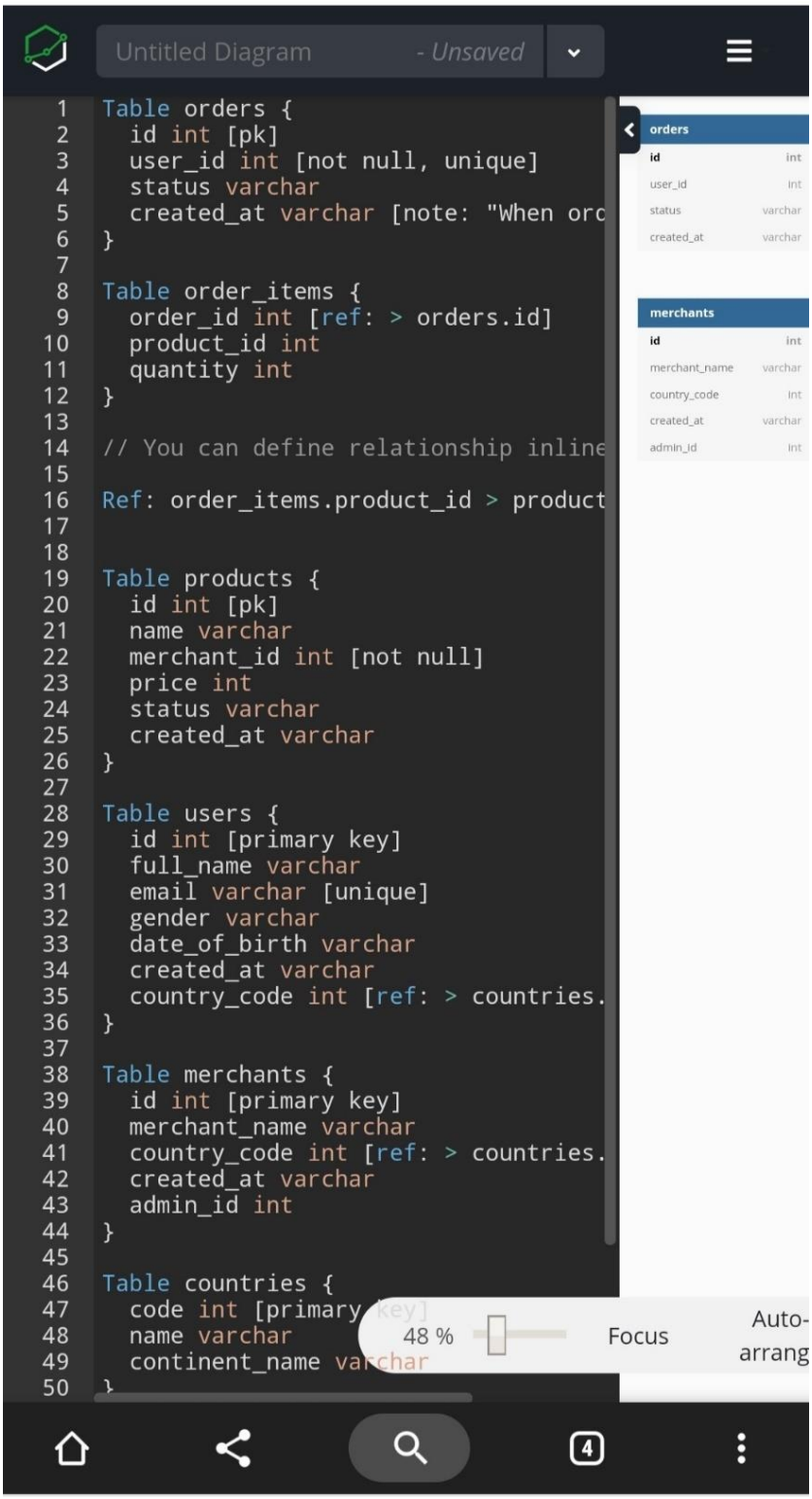

Рис. 10. dbdiagram.io мобильная версия.

На основе проведенного анализа можно сделать несколько выводов касающихся крупнейших представленных онлайн сервисов по проектированию баз данных:

- сервисы в должной мере не адаптированы под использование на мобильных устройствах;
- у сервисов отсутствуют мобильные приложения для мобильных операционных систем.

Таким образом, на рынке специализированного программного обеспечения не представлены решения, ориентированные на проектирование баз данных, на мобильном устройстве под операционной системой Android.

1.2 Инструментальные средства разработки мобильного приложения

Разработка клиентской части в формате мобильного приложения целиком либо частично возможна лишь на небольшом количестве языков программирования. Выбор языка программирования – это самый важный этап в разработке, так как именно он определит дальнейшую парадигму развития любого проекта. Проведем сравнительный анализ некоторых языков программирования, отвечающих задаче работы.

Самым популярным языком в мобильной разработке является Java. Этот язык также является самым популярным в мире относительно всех остальных языков по оценке TIOBE [32]. Подавляющее большинство образовательной документации и открытых исходных кодов приложений для Android написано именно на Java. Этот язык является фундаментальным и кроссплатформенным. Основные недостатки Java – низкое, относительно других языков, быстродействие и высокое потребление оперативной памяти [36].

Еще один из языков, позволяющий разрабатывать приложения под Android, является C#. Он был разработан компанией Microsoft и основан на языке Java. Опираясь на практику таких языков программирования как Java, C++,

Pascal и др. C# был лишен некоторых моделей, зарекомендовавших себя проблематичными [43]. Разработка на C# под Android стала возможна благодаря компании Xamarin, которая создала одноименный фреймворк. Этот фреймворк позволяет создавать кроссплатформенные приложения, используя при этом один и тот же код. Xamarin позволяет создавать приложение одновременно для таких операционных систем как: iOS, Android, Windows и macOS [45]. Не смотря на большое количество преимуществ, Xamarin имеет несколько весомых недостатков. Например, большой вес приложений и сложности с интеграцией. Также стоит сказать о том, что при разработке на Xamarin потребуется писать платформо-зависимый слой кода для приложений с нативным интерфейсом. Поэтому потребуется базовые знания нативных языков программирования – Java и Objective C или Swift [35].

На данный момент Python не поддерживается Android для написания нативных[1] приложений. Однако, существуют инструменты, позволяющие скомпилировать код на Python в байт код, интерпретируемый JVM. Один из таких инструментов – библиотека с открытым исходным кодом Kivy. Эта библиотека позволяет писать один код сразу для macOS, OS Linus, Windows, Android и iOS. Но не смотря на все плюсы, Java Virtual Machine официально не поддерживает Python и как следствие вызывает большое количество ошибок. Такой код очень трудно поддерживать и вести непрерывную интеграцию [28].

[1] Нативные приложения (native application) — это прикладная программа, которая была разработана для использования на специфической платформе или устройстве. Главное преимущество нативных приложений – то, что они оптимизированы под конкретные операционные системы, а значит работают корректно и быстро. Также они имеют доступ к аппаратной части устройств, то есть могут использовать в своем функционале камеру смартфона, микрофон, акселерометр, геолокацию, адресную книгу, плеер и т.д. Термин «нативное приложение» часто упоминается в контексте мобильной разработке, поскольку мобильный софт традиционно пишется, чтобы работать на определенной аппаратной платформе. Нативное приложение устанавливается на мобильном устройстве непосредственно производителем или может быть загружено со сторонних источников (таких как Google Play или Apple App Store) [41].

Языки для разработки веб приложений с недавнего времени стали поддерживать мобильную разработку. Так, например, благодаря таким инструментам как PhoneGap и Apache Cordova с помощью HTML, CSS и JavaScript можно разработать мобильное приложение с простым функционалом [29]. Основной недостаток данных инструментов – их ограниченный функционал. Если же посмотреть на React native – разработанный компанией The Facebook фреймворк для JavaScript, то откроется намного больше функций. React native это библиотека, которая позволяет разрабатывать кроссплатформенные приложения на JavaScript [35]. Но React native лишен одного из самых главных инструментов – качественных библиотек. Со временем сложилась такая ситуация, что при разработке проектов на JavaScript используются очень большое количество сторонних библиотек и без них JavaScript крайне ограничен.

Еще одним из языков, поддерживающих мобильную разработку, является Lua. Этот язык старше чем Java, менее популярный, но тем не менее востребованный. Инструмент с помощью которого ведется разработка на Lua называется Corona SDK. Это фреймворк, позволяющий разрабатывать кроссплатформенные приложения, чаще – игры [14]. Именно благодаря простой библиотеке, позволяющей создавать игры, Lua является до сих пор востребованным языком программирования.

Компания Google Inc, владеющая ОС Android, предоставляет сторонним разработчикам две среды разработки: SDK – предназначена для работы с Java и NDK – предназначена для работы с нативными языками C и C++. Следует оговориться о подходе разработки с помощью NDK – полноценное приложение разработать таким путём невозможно. NDK предназначена исключительно для написания встраиваемых модулей и собственных библиотек для подключения их к основному коду приложения[31]. Модули, разработанные с помощью NDK имеют очень высокую производительность так как она используют нативные языки.

Новейшим языком, который поддерживаемым JVM стал Kotlin - разработанный в 2011 и выпущенный в 2017 году компанией JetBrains. Kotlin полностью совместим с Java и имеет официальную поддержку и документацию от Google. Kotlin позиционирует себя, как язык, который сможет со временем заменить Java. Язык был разработан с упором на повышение производительности [37].

Также существует еще один способ разработки мобильных приложений для операционной системы Android. Flutter – это SDK от компании Google LLC для разработки кроссплатформенных мобильных приложений под Android и iOS. Flutter приложения пишутся с использованием языка Dart. Главное преимущество платформы Dart в том, что он использует «горячую перезагрузку» это такой процесс, при котором изменение исходного кода применяется в скомпилированном приложении, без необходимости перезагрузки. Dart написан на языке C++, графическая библиотека Google Skia помогает обеспечить движку низкоуровневый рендеринг. Также платформозависимымые SDK под Android и iOS могут взаимодействовать с платформой Dart.

Так как требуется разрабатываемое приложение планируется использовать только для операционной системы Android, то исчезает необходимость в выборе тех языков, чей упор сделан на кроссплатформенность. Проанализировав каждый из языков программирования, целесообразно выделить Java. Этот язык на данный момент является самым используемым в корпоративной разработке высоконагруженных систем и обеспечивает высокую надёжность кода.

Также при разработке любого приложения выбор интегрированной среды разработки, в которой будет происходить разработка с использованием непрерывной интеграцией является основополагающим фактором, который позволит выбрать дальнейший вектор развития разрабатываемого продукта. Рассмотрим и проведем сравнительный анализ некоторых сред разработок, с помощью которых возможно производить разработку приложений для мобильных операционных систем.

Microsoft Visual Studio [46] – это набор продуктов от Microsoft, содержащая интегрированную среду разработки ПО и другие инструментальные средства. Данные продукты дают возможность вести разработку консольных приложений и приложений с графическим интерфейсом, в том числе с полной поддержкой Windows Forms, а также веб-приложения, веб-сайты как в встроенном, так и в управляемом кодах для всех платформ, поддерживаемых Xbox, Windows, Windows Mobile .NET Framework, Windows Phone.NET Compact Framework и Silverlight. Visual Studio включает в себя редактор кода с поддержкой IntelliSens. Встроенный отладчик может работать как отладчик машинного уровня, так и отладчик уровня исходного кода. Другие встраиваемые инструменты содержат в себе редактор форм для создания графического интерфейс, веб-редактор, конструктор дизайна классов и конструктор дизайна схемы базы данных. Visual Studio позволяет подключать и создавать плагины для расширения функционала. Присутствует интегрированная система контроля версий кода (Git, scm и т.д). Также обладает возможностью добавления новых инструментов или инструментов для других сторон процесса разработки ПО. Visual Studio имеет возможность разработки кроссплатформенных мобильных приложений благодаря плагину от компании Xamarin [14].

IntelliJ IDEA [19] – интегрированная среда разработки ПО для обширного количества языков программирования, таких как Python, Kotlin, Java, разработанная компанией JetBrains. Первая версия этой IDE приобрела статус первой среды для Java с наиболее широким подбором интегрированных инструментов для рефакторинга исходного кода, которые дают разработчикам перестраивать исходные данные ПО. Также IntelliJ IDEA содержит инструментарий для разработки графического интерфейса пользователя (GUI). Среда совместима с большинством свободных инструментов и плагинов, таких как Apache Maven, Gradle, Apache Ant, Junit и др [20]. На данный момент IntelliJ IDEA поддерживает более 20 различных языков программирования и имеет возможность работать в SDK Android, что в свою очередь позволяет создавать мобильные приложения для

операционной системы Android. Основным недостатком этой среды разработки является производительность. Компиляция, тестирование, сборка и прочие процессы занимают значительное количество времени относительно других IDE.

Eclipse [18] – свободная интегрированная среда разработки модульных кроссплатформенных приложений. Развивается и поддерживается компанией Eclipse Foundation. Eclipse является кроссплатформенной IDE так как она написана на Java, за исключением библиотеки Standard Widget Toolkit, которая разрабатывается для многих платформ. Библиотека Standard Widget Toolkit эксплуатируется вместо стандартной библиотеки Swing на Java. Она полностью базируется на операционной системе, что гарантирует быстродействие и естественный внешний вид интерфейса пользователя, но также вызывает на различных платформах проблемы совместимости и устойчивости ПО. Eclipse совместима с большим количеством самых популярных открытых инструментов для разработки ПО [22]. Также это среда разработки поддерживает и написание приложений под ОС Android на языке Java и при установленном соответствующем плагине на языке Kotlin. Существенными недостатками Eclipse являются: нехватка официальной документации и отсутствие единого сообщества разработчиков, что в свою очередь затруднит разработку независимого продукта.

NetBeans – это свободная среда разработки ПО на таких языках как: Python, JavaScript, Java, C++, C и др. Несмотря на то, что NetBeans имеет поддержку со стороны компании Oracle, её разработка ведётся компанией NetBeans Org и сообществом разработчиков. В NetBeans присутствует профилирование кода и его рефакторинг. Для разработки ПО в NetBeans требуется наличие Sun JDK или J2EE SDK [26]. Среда разработки. Также NetBeans поддерживает разработку приложений для мобильных ОС при установке соответствующего SDK. Также для этой среды разработки существует плагин для поддержки языка программирования Kotlin. Основная проблема данной IDE возникает именно при её эксплуатации непосредственно для создания мобильных приложений. Большинство плагинов для этой IDE которое созданы для разработки мобильных приложений

имеют сильно ограниченный функционал. Данная проблема в совместимости с низкой производительностью NetBeans делает разработку ПО для мобильных платформ крайне затруднительной.

Android Studio [12] – это интегрированная среда разработки для работы с платформой Android. После выпуска первой стабильной версии Android Studio окончило свое существование поддержка Android Development Tools для IDE Eclipse. Android Studio, основанная на программном обеспечении IntelliJ IDEA от компании JetBrains, – официальное средство разработки Android приложений. Данная среда разработки доступна для Windows, OS X и Linux. На ежегодной конференции Google I/O, Google анонсировал язык Kotlin, используемый в Android Studio как официальный язык программирования для платформы Android в добавление к Java и C++ [36]. На данный момент Android Studio является уникальным в своем роде продуктом, который позволяет вести непрерывную интеграцию при разработке мобильных приложений и имеющий за собой официальную документацию. Среди всех IDE, Android Studio зарекомендовала себя как лучшая среда разработки для создания мобильных приложений, функционирующих на базе операционной системы Android. Данная IDE поддерживает рефакторинг кода, имеет встроенный сборщик проектов Gradle, позволяет компилировать и устанавливать apk файлы непосредственно с физическим устройством в процессе разработки ПО.

На основе проведенного сравнительного анализа, среди всех представленных интегрированных сред разработки стоит выделить Android Studio, так как разработка приложения будет производиться для операционной системы Android. Выбранная IDE обладает всеми необходимыми инструментами для разработки приложения с применением непрерывной интеграции: система контроля версий исходного кода (VCS), отладка программного обеспечения на физических и виртуальных устройствах, модульность, официальная поддержка языка программирования Java, а также использование встроенных сборщиков проектов и др. Intellij Idea обладает теми же преимуществами что и Android Studio при

установки только необходимых плагинов и при этом потребляет гораздо меньше ресурсов, чем Android Studio. Именно производительность является важнейшим фактором при выборе между этими средами разработки, поэтому **Intellij Idea** была выбрана в качестве основной среды разработки для разработки клиентской части приложения[40, 36].

После выбора языка программирования и среды разработки необходимо определиться с перечнем технологий, которые понадобятся для функционирования клиентской части приложения. Так как передача данных между клиентом и сервером реализована с помощью архитектурного стиля REST, то для удобного управления объектами, которые передаются по HTTP запросам следует использовать какую-либо библиотеку для работы с HTTP запросами. Рассмотрим HTTP библиотеку Retrofit 2.

Retrofit 2 – это потокобезопасный HTTP клиент для Java и Android разработанный компанией Square. Retrofit является незаменимым инструментом в клиент-серверных приложениях для работы с API[2]. Retrofit позволяет сделать полноценный REST-клиент, который может выполнять POST, GET, PUT, DELETE. UPDATE Для обозначения типа и других аспектов запроса используются аннотации. Например, для того чтобы обозначить, что требуется GET запрос, необходимо написать перед методом «@GET», для POST запроса «@POST», и так далее. В скобках к типу запроса устанавливается целевой адрес.

После того как произведен сравнительный анализ и выбор технологий для разработки клиентской части, необходимо перейти к сравнительному анализу технологий и языка программирования для разработки серверной части приложения. Обращаясь к рейтингу от TIOBE, можно отметить, что за всё время существования данного рейтинга, лидерами зачастую являлись языки C++ и Java. Но так как в рамках приложения «EasyBox» предполагается, что сервер не является высоконагруженной

[2] API (программный интерфейс приложения, интерфейс прикладного программирования) (англ. Application programming interface, API) — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

системой с необходимостью в высокой производительности и безопасности, то следует обратить внимание на набирающий популярность язык Python.

Pyhton – это высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций. Так как серверная часть приложения содержит в себе мало бизнес-логики относительно крупных проектов, то Python является корректным выбором для разработки серверной части.

Сервер приложения постоянно принимает HTTP запросы и на их основе производит манипуляции с данными, которые находятся в базе данных. Сервер размещен на условно бесплатном ресурсе «pythonanywhere» и в качестве СУБД используется SQLite – библиотека на языке C, которая реализует небольшой, быстрый, автономный, высоконадежный, полнофункциональный механизм базы данных SQL. SQLite – самый распространенный в мире движок баз данных. SQLite встроен во все мобильные телефоны и большинство компьютеров и поставляется внутри множества других приложений, которые люди используют каждый день. Сервер функционирует за счёт фреймворка flask-sqlalcemhy, который, в свою очередь, является симбиозом микрофреймворка flask, который создан для создания веб-приложений, и ORM[3] библотеки SQLAlcmemy, которая предназначена для работы с реляционными СУБД[29, 19].

1.3 Формализованное описание технического задания

1. Общие сведения.

1.1. Название организации-заказчика.

Уральский Государственный Педагогический Университет кафедра информационно-коммуникационных технологий в образовании.

[3] ORM (англ. Object-Relational Mapping, рус.объектно-реляционное отображение, или преобразование)— технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

1.2. Название продукта разработки (проектирования).

Мобильное приложение под управлением операционной системы Android «EasyBox».

1.3. Назначение продукта.

Предназначается для генерации кода Data Definition Language (DDL) (язык описания данных) на языке SQL, с возможностью выбора SQL диалекта. Генерация кода происходит на основе заданных базовых элементов баз данных – таблиц, полей, связей. Приложение должно функционировать на мобильном устройстве под управлением операционной системы Android. [12, 14, 22]

1.4. Плановые сроки начала и окончания работ.

В соответствии с графиком учебного процесса с 10.01.2019 по 20.05.2019.

2. Характеристика области применения продукта.

2.1. Процессы и структуры, в которых предполагается использование продукта разработки.

Полиморфное проектирование баз данных для заданных систем управления базами данных.

2.2. Характеристика персонала (количество, квалификация, степень готовности).

2.2.1. Разработчик – знание Java или Kotlin, опыт работы не менее 1 года. Опыт разработки мобильных приложений в интегрированной среде разработки Intellij IDEA или Android Studio

2.2.2. Инженер-разработчик баз данных – базовые знания SQL и его диалектов (PostgreSQL, SQLite). Опыт разработки БД от 1 года.

3. Требования к продукту разработки.

3.1. Требования к продукту в целом.

Возможность регистрации пользователей по электронной почте и паролю в системе. Вход с помощью электронной почты и пароля. Создание модели базы данных, включающей в себя создание, удаление, редактирование, обновление

таблиц, полей, связей. Наличие SQL генератора в такие СУБД, как: PostgreSQL, SQLite.

3.2. Аппаратные требования.

Смартфон или планшетный ПК с поддержкой стандарта передачи данных не ниже, чем 3G для доступа в сеть интернет.

3.3. Указание системного программного обеспечения (операционные системы, браузеры, программные платформы и т.п.).

Операционная система Android не ниже версии 7.1.

3.4. Указание программного обеспечения, используемого для реализации.

Android SDK, Intellij Idea 2019.3.1, Java 8, Photoshop CC, Git, Ramus Educational, Visual Studio Code, Python 3.

3.5. Для сетевых систем – особенности реализации серверной и клиентской частей.

Серверная часть должна быть реализована с использованием Python 3, в качестве СУБД следует использовать SQLite. Сервер должен работать на фреймворке flask-sqlalchemy и взаимодействовать с клиентов посредством HTTP запросов по архитектуре REST. Сервер должен быть размещён на условно бесплатном хостинге «pythonanywhere» Клиентская часть должна быть реализована в качестве apk приложения для операционной системы Android 7.0 и выше. Для обмена состояния между клиентом и сервером следует применить на стороне клиента фреймворк «retrofit 3.0».

3.6. Форматы входных и выходных данных

На выход подается управляющая информация (тап по экрану) для создания модели базы данных и всех её составляющих (таблиц, полей, связей). Выходные

данные представляются в виде редактируемого SQL кода (текста) на создание созданной модели базы данных для выбранной пользователем СУБД.

3.7. Источники данных и порядок их ввода в систему (программу), порядок вывода, хранения.

Источником данных выступают:

- сервер, работающий на системе «pythonanywhere», конфиденциальные данные пользователей хранятся в зашифрованном виде в СУБД:
- данные пользователя.

3.8. Порядок взаимодействия с другими системами, возможности обмена информацией.

Мобильное приложение постоянно взаимодействует с сервером, отправляя HTTP запросы по архитектуре REST в формате JSON.

3.9. Меры защиты информации.

Данный программный продукт является open-source реализацией, его исходный код открыт и доступен в репозитории Github.

4. Требования к пользовательскому интерфейсу.

4.1. Общая характеристика пользовательского интерфейса.

Высококонтрастное цветовое решение на основе холодных тонов для комфортного использования приложения при естественном освещении.

4.2. Размещение информации на экране, дизайн экрана.

Дизайн экрана и размещение элементов должно советовать представленным макетам:

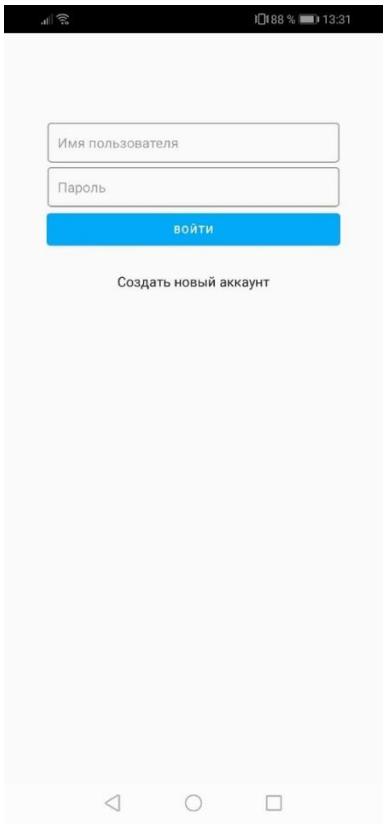

Рис. 11. Авторизация

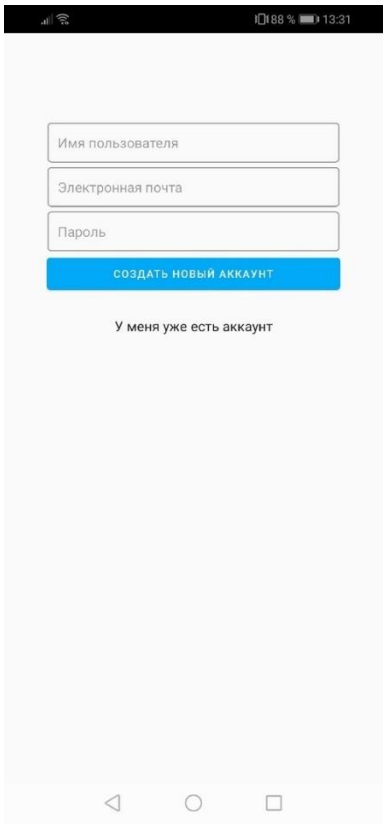

Рис. 12. Регисрация

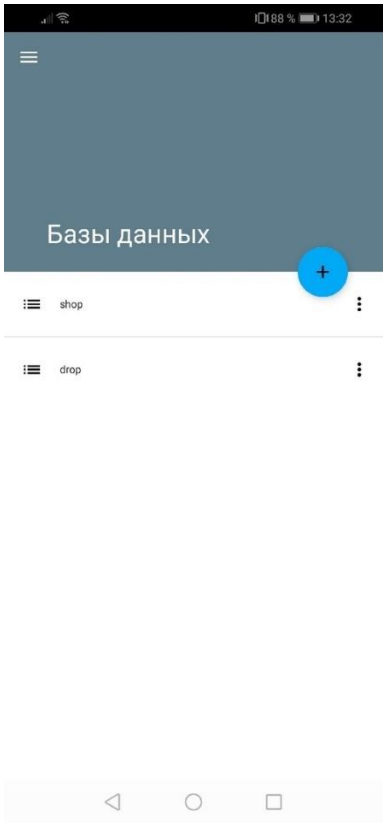

Рис. 13. Базы данных

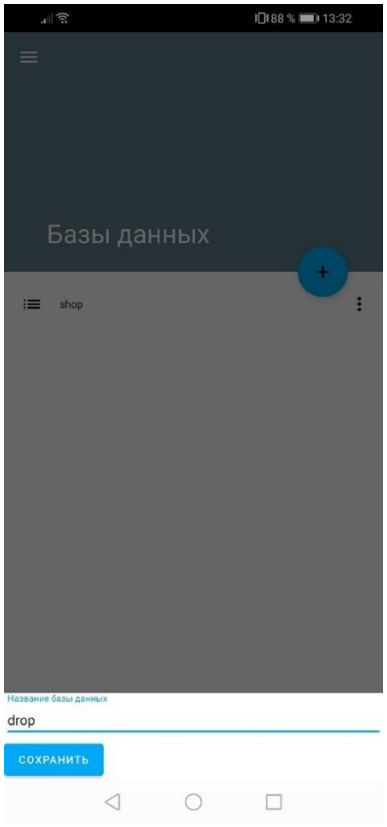

Рис. 14. Создание/редактирование БД

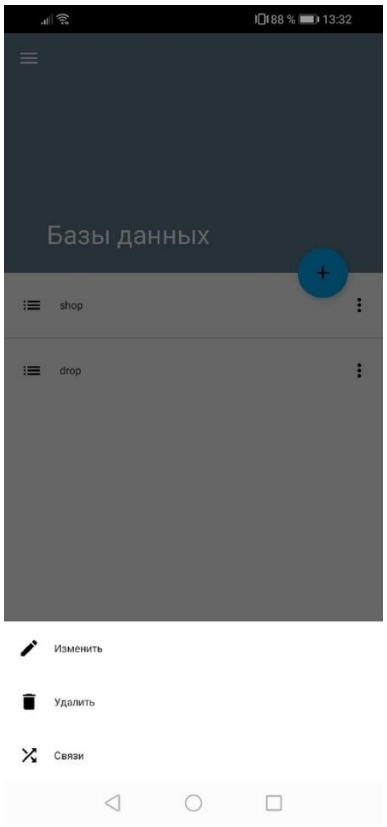

Рис. 15. Действия над объектом

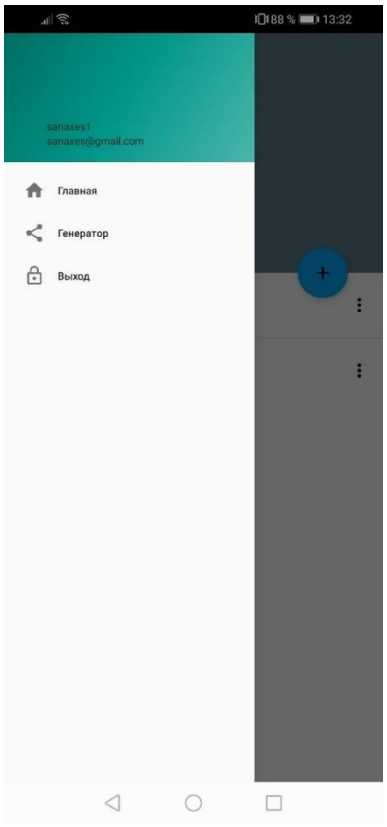

Рис. 16. Боковое меню

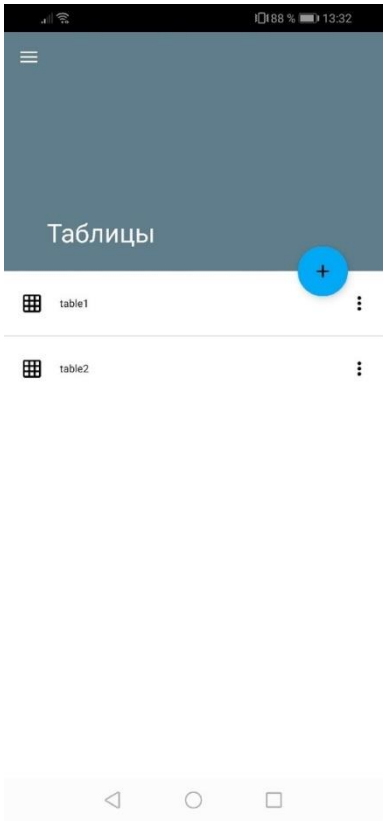

Рис. 17. Таблицы

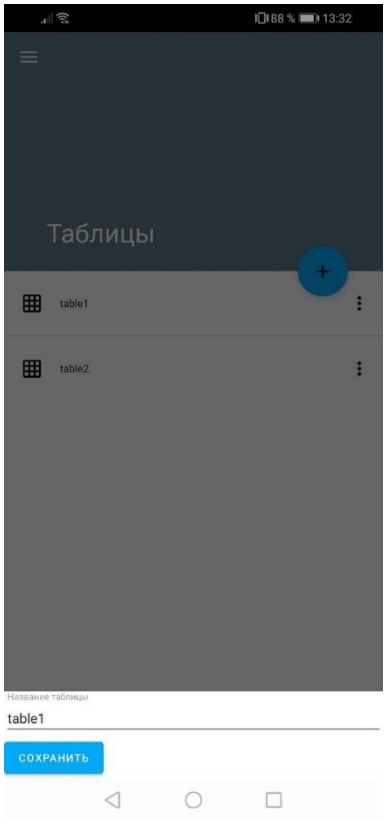

Рис. 18. Создание/редактирование таблицы

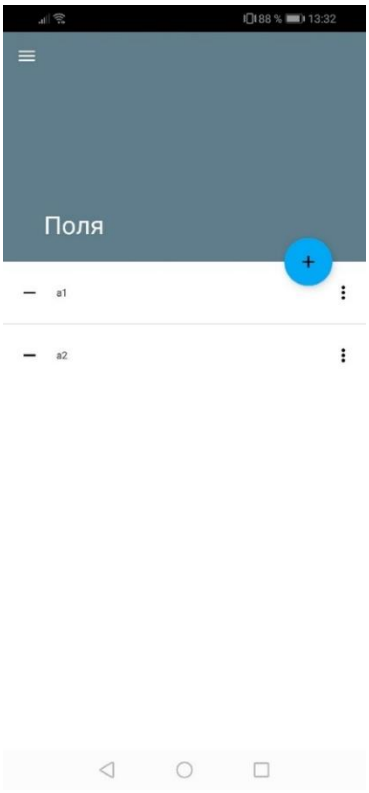

Рис. 19. Поля

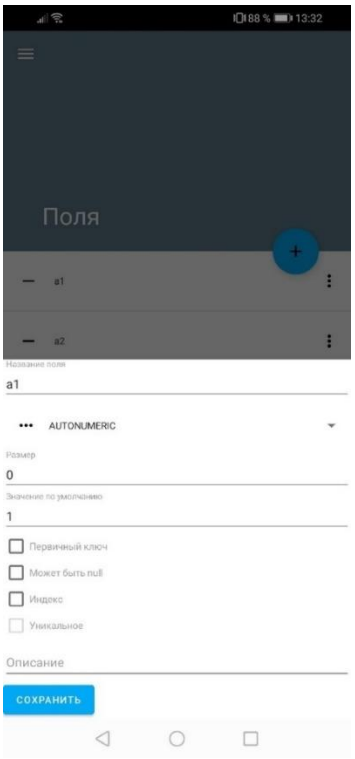

Рис. 20. Создание/редактирование поля

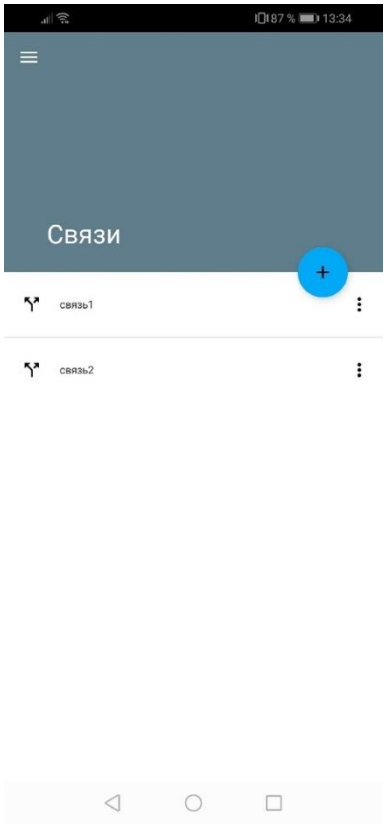

Рис. 21. Связи

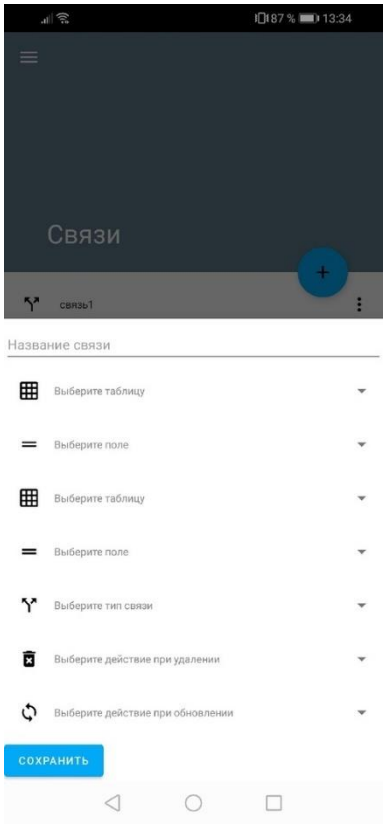

Рис. 22. Редактирование/создание связи

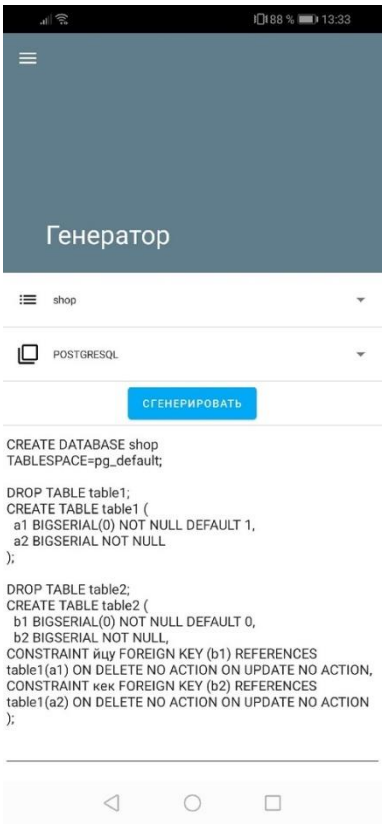

Рис. 23. Генератор кода

4.3. Особенности ввода информации пользователем, представление выходных данных.

Пользователь вводит информацию прикосновением по сенсорному экрану устройства. Выходные данные представляются в виде окна с редактируемым текстом, в котором находится SQL код.

5. Порядок сдачи-приемки продукта.

В соответствии с графиком выполнения выпускной квалификационной работы.

Глава 2. Реализация мобильного приложения EasyBox

2.1 Информационные и функциональные модели приложения EasyBox

В разработанном продукте представлено более 30 классов и интерфейсов на клиентской и серверной части, а также 7 экранов взаимодействия с приложением. Поскольку разработанная информационная система является сложным технологическим продуктом, то требуется вести сопроводительную техническую документацию по ходу увеличения объема программного кода. При разработке комплексных информационных систем, имеющих потенциал последующей модернизации и изменений, следует использовать информационные модели, для того чтобы облегчить порог вхождения разработчикам в данный продукт.

Разберем более подробно информационную модель информационной системы «EasyBox». Для начала необходимо рассмотреть архитектурную информационную модель (Рис. 24) представленного решения. На представленной схеме отображено как компоненты системы взаимодействуют друг с другом:

- в облачном сервисе «pythonanywhere» функционирует сервер приложения на основе фреймворка «flask-sqlalchemy» и выполняет взаимодействие с базой данных, расположенной также в облаке, под управлением СУБД SQLite.
- обмен данными происходит по архитектурному стилю REST посредством HTTP запросов от устройств пользователей (клиентов) к серверу.

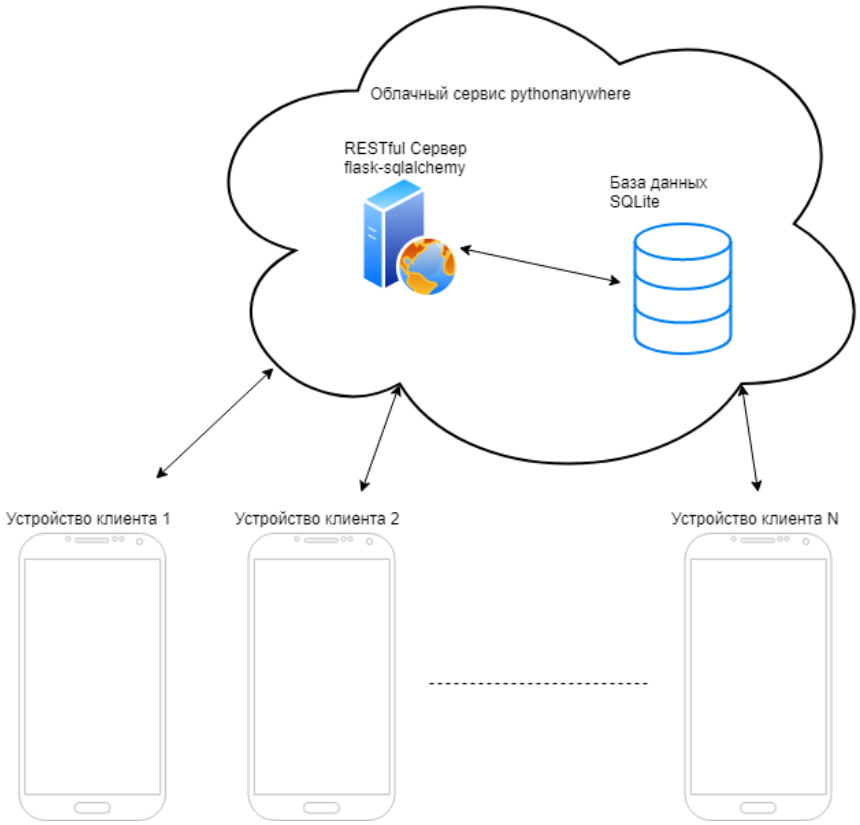

Рис. 24. Схема архитектурной модели

Также следует рассмотреть функциональную модель. Поскольку клиентская часть реализована в качестве мобильного приложения, целесообразно будет отразить на функциональной диаграмме IDEF0 взаимодействие между Activity[4] приложения. Диаграмма уровня A0 (Рис. 25) отражает что приложение принимает на вход сетевое соединение и HTTP запрос к серверу, на выходе приложение предоставляет результат HTTP запроса и сгенерированный SQL код. В качестве управления приложение принимает ГОСТы по разработке программного обеспечения, а в качестве управления пользователя и разработчика. На диаграмме декомпозиции блока A0 (Рис. 26) представлены два блока: приложение и интегрированная среда разработки. Блок интегрированной среды разработки яв-

[4] Activity — это компонент приложения, который выдает экран, и с которым пользователи могут взаимодействовать для выполнения каких-либо действий.

ляется самостоятельной системой и не требует рассмотрения декомпозиции. Декомпозиция блока A1 приложение (Рис. 27) содержит в себе информацию о взаимодействии между всеми Activity приложения.

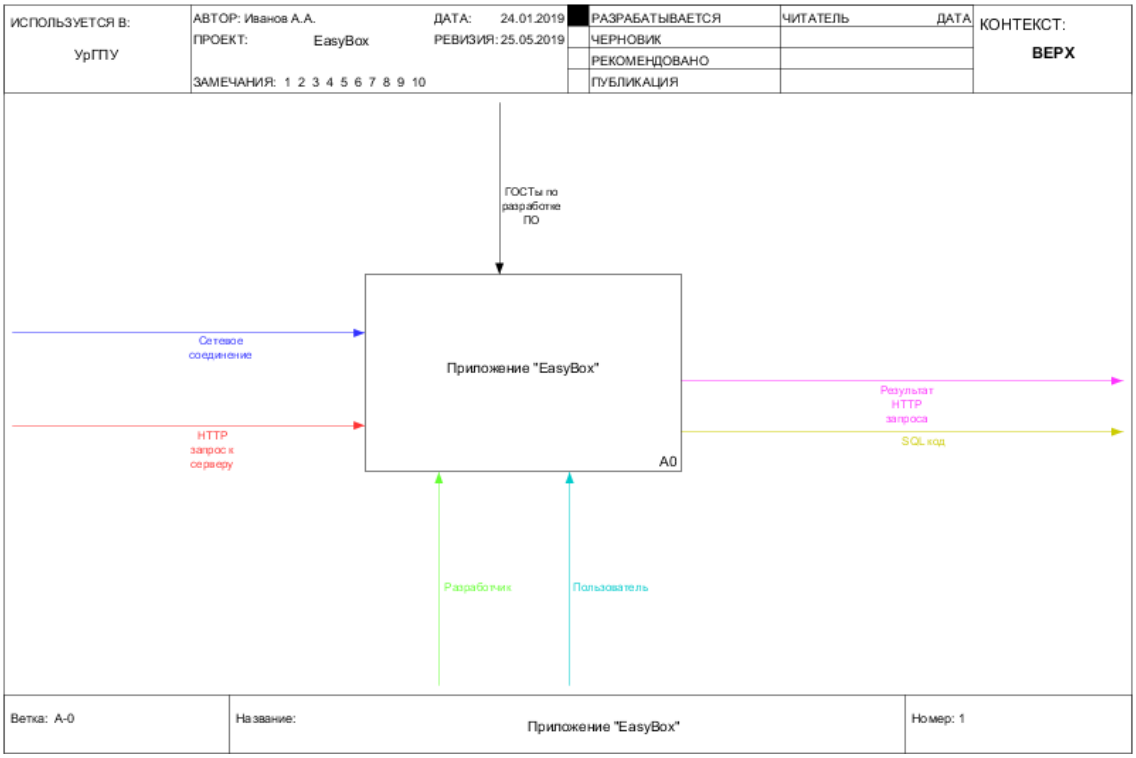

Рис. 25. IDEF0 Диаграмма A0

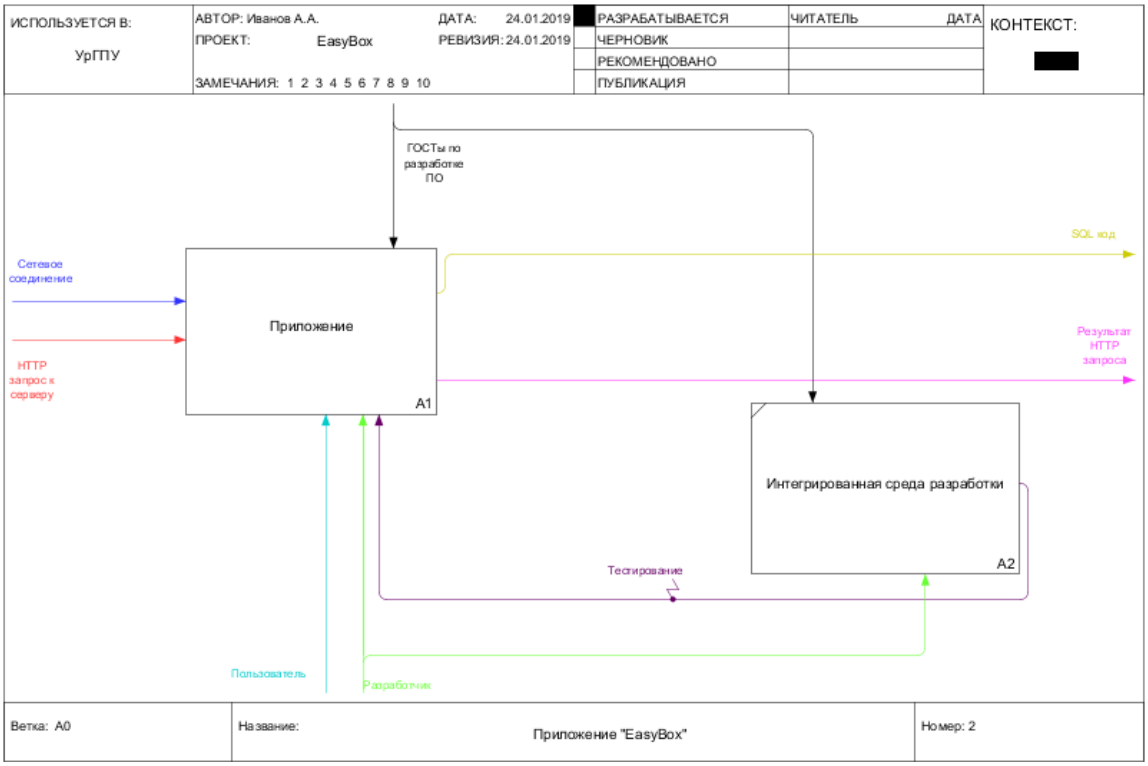

Рис. 26. IDEF0 Диаграмма декомпозиции блока A0

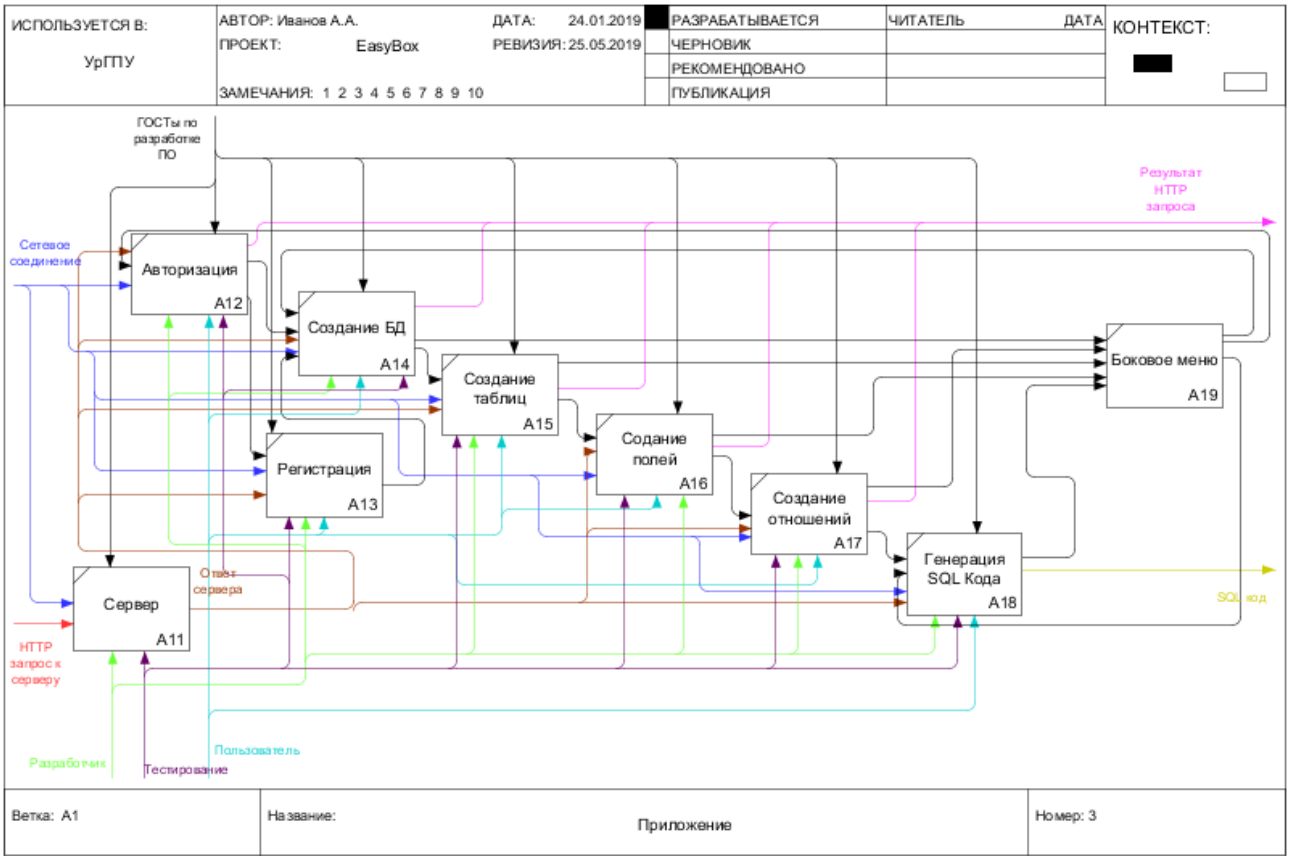

Рис. 27. IDEF0 Диаграмма декомпозиции блока A1

Рассмотрим диаграмму, отражающую структуру пакетов исходного кода (Рис. 28). Весь исходный код находится внутри трехуровневого пакета «ru.sanaxes.easybox.*», такое наименование считается общепринятым, где первый уровень «ru» указывает на страну в которой производился продукт, второй уровень «sanaxes» указывает на имя собственное – владелец продукта, а третий уровень «easybox» указывает на название продукта. Рассмотрим пакеты содержащиеся в пакете «ru.sanaxes.easybox.*».

- controllers содержит в себе классы, которые указывают маппинг[5] для CRUD операций над объектами системы по архитектуре REST;
- domains содержит в себе классы-представления базовых сущностей в формате POJO, на которых основана информационная система (класс пользователь, класс таблица, и т.д.);

[5] Маппинг – это определение соответствия данных между потенциально различными семантиками одного объекта или разных объектов.

- generators содержит в себе классы для генерации SQL кода под различные системы управления базами данных;
- services содержит в себе классы с бизнес-логикой. Также классы в пакете services отвечают за выполнение HTTP запросов на CRUD операции с помощью фреймворка Retrofit;
- ui содержит в себе классы, отвечающие за пользовательский интерфейс (Activity классы, пользовательские View и т.д);
- utils содержит в себе вспомогательные классы-утилиты.

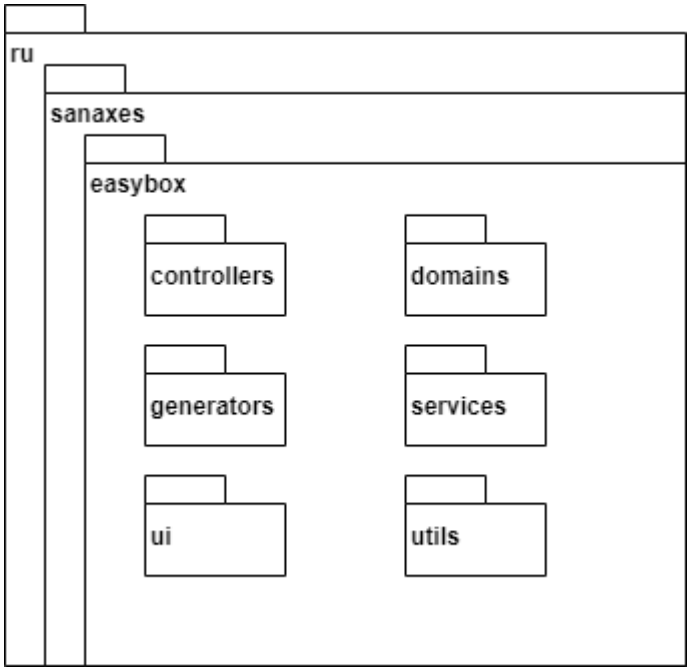

Рис. 28. Диаграмма структуры пакетов

Рассмотрим диаграммы HTTP запросов по архитектуре RESTful API на CRUD операции для базовых сущностей.

На диаграмме HTTP запросов для пользователя (Рис. 29) видно, из чего состоит сущность User: id – уникальный идентификатор, username – имя пользователя, email – электронная почта, password_hash – пароль в зашифрованном виде, databases – базы данных пользователя. Операции доступные для сущности «пользователь»:

- создание – POST запрос по URI – «/api/v1/users»;
- получение пользователя по уникальному идентификатору – GET запрос по URI – «/api/v1/users/{userId}», где userId – идентификатор пользователя;

- получение пользователя по имени пользователя – GET запрос по URI – «/api/v1/users/{username}», где username –имя пользователя.

Возможные ответы сервера– неверный запрос (код 400 Bad Request), повторный пользователь (код 409 Conflict), пользователь создан (код 201 Created).

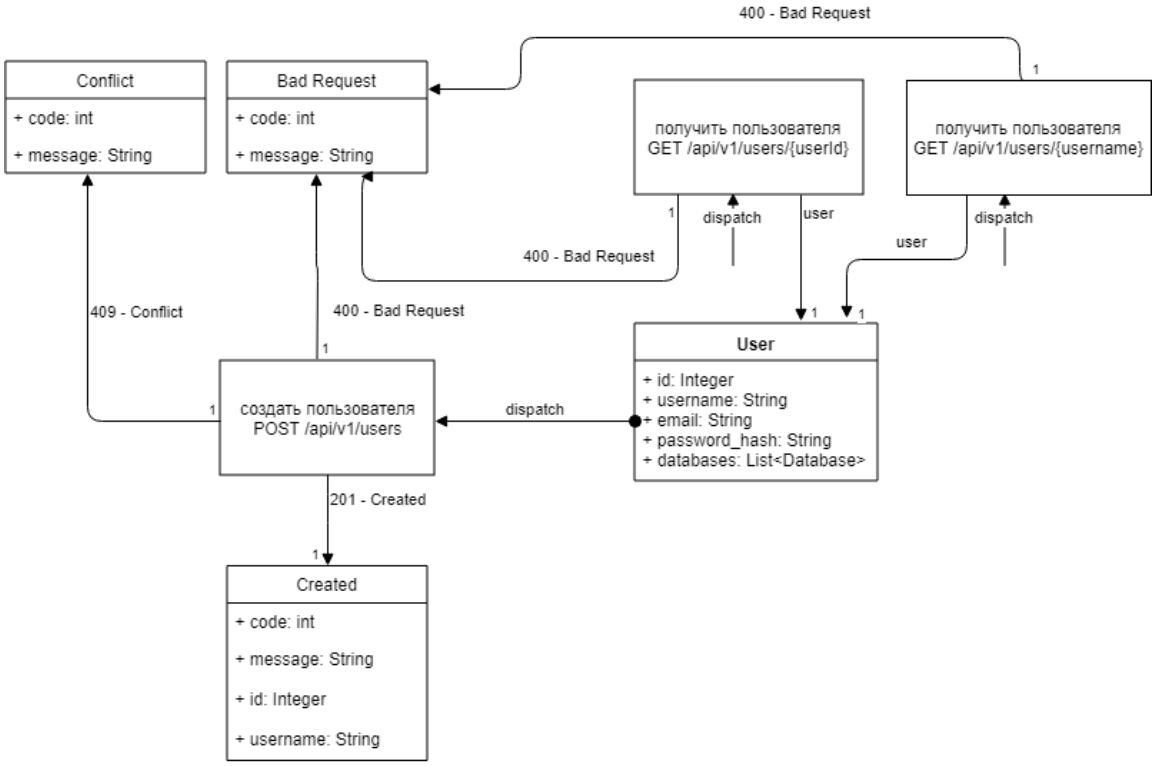

Рис. 29. Диаграмма HTTP запросов для пользователя

На диаграмме HTTP запросов для базы данных (Рис. 30) видно, из чего состоит сущность Database: user_id – уникальный идентификатор пользователя, id – уникальный идентификатор базы данных, name – наименование, tables –таблицы, relationships –связи. Операции доступные для сущности «база данных»:

- создание – POST запрос по URI – «/api/v1/database»;
- получение всех баз данных – GET запрос по URI – «/api/v1/database»;
- получение базы данных по уникальному идентификатору – GET запрос по URI – «/api/v1/database/{databaseId}», где databaseId – идентификатор базы данных;
- обновление базы данных по уникальному идентификатору – PUT запрос по URI – «/api/v1/database/{databaseId}»;

- удаление базы данных по уникальному идентификатору – DELETE запрос по URI – «/api/v1/database/{databaseId}».

Возможные ответы сервера– неверный запрос (код 400 Bad Request), база данных создана (код 201 Created), база данных обновлена (код 200 OK), база данных удалена (код 204 No content)

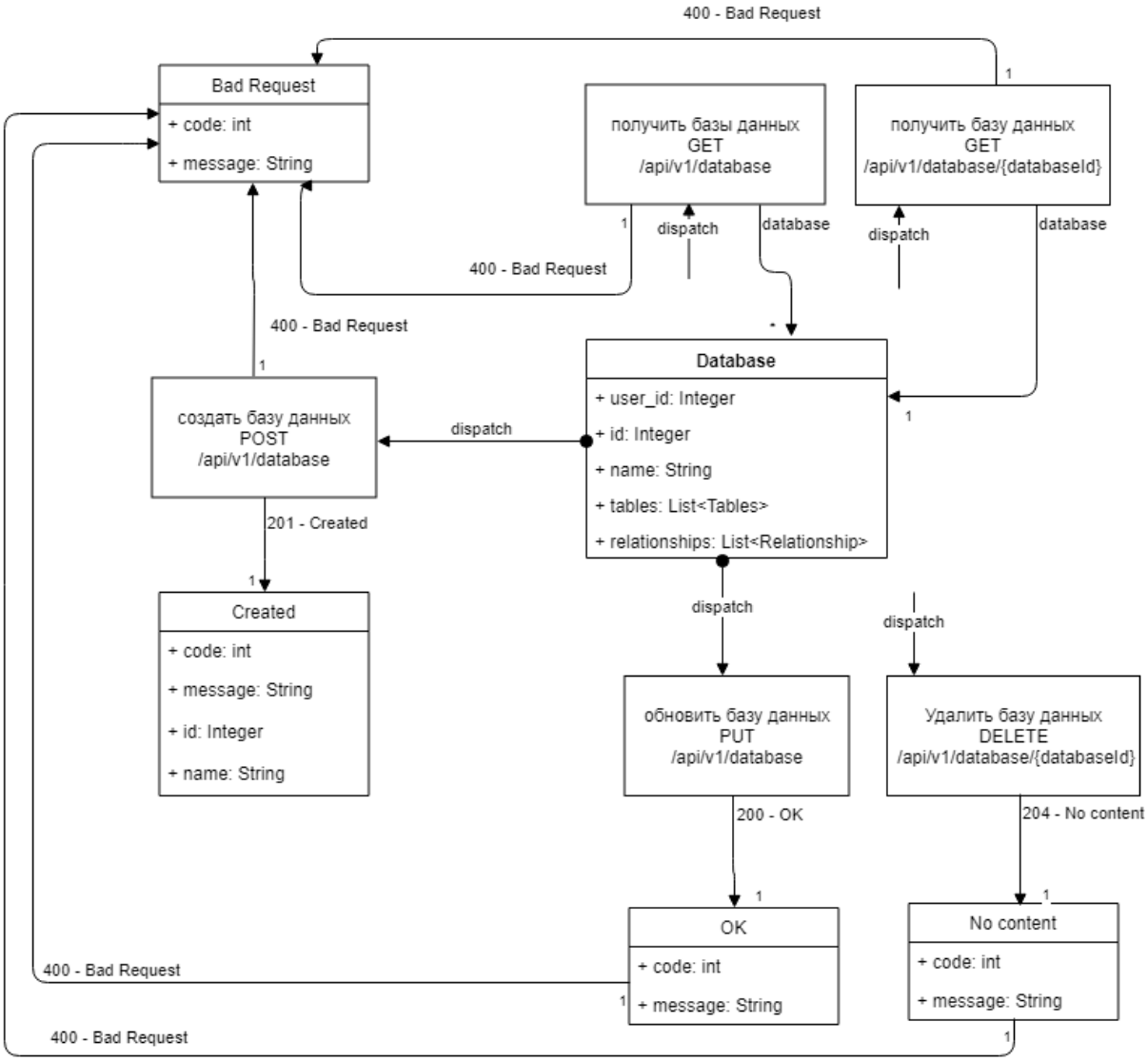

Рис. 30. Диаграмма HTTP запросов для базы данных

На диаграмме HTTP запросов для таблицы (Рис. 31) видно, из чего состоит сущность Table: user_id – уникальный идентификатор пользователя, database_id – уникальный идентификатор базы данных, id – идентификатор таблицы, name – наименование таблицы, fields –поля. Операции доступные для сущности «таблица»:

- создание – POST запрос по URI – «/api/v1/database/{databaseId}/table», где databaseId – идентификатор базы данных;
- получение всех таблиц базы данных – GET запрос по URI – «/api/v1/database/{databaseId}/table»;
- получение таблицы по уникальному идентификатору базы данных и таблицы – GET запрос по URI – «/api/v1/database/{databaseId}/table/{tableId}», где tableId – идентификатор таблицы;
- обновление таблицы по уникальному идентификатору базы данных и таблицы– PUT запрос по URI – «/api/v1/database/{databaseId}/table/{tableId}»;
- удаление базы данных по уникальному идентификатору базы данных и таблицы– DELETE запрос по URI – «/api/v1/database/{databaseId}/table/{tableId}».

Возможные ответы сервера– неверный запрос (код 400 Bad Request), таблица создана (код 201 Created), таблица обновлена (код 200 OK), таблица удалена (код 204 No content).

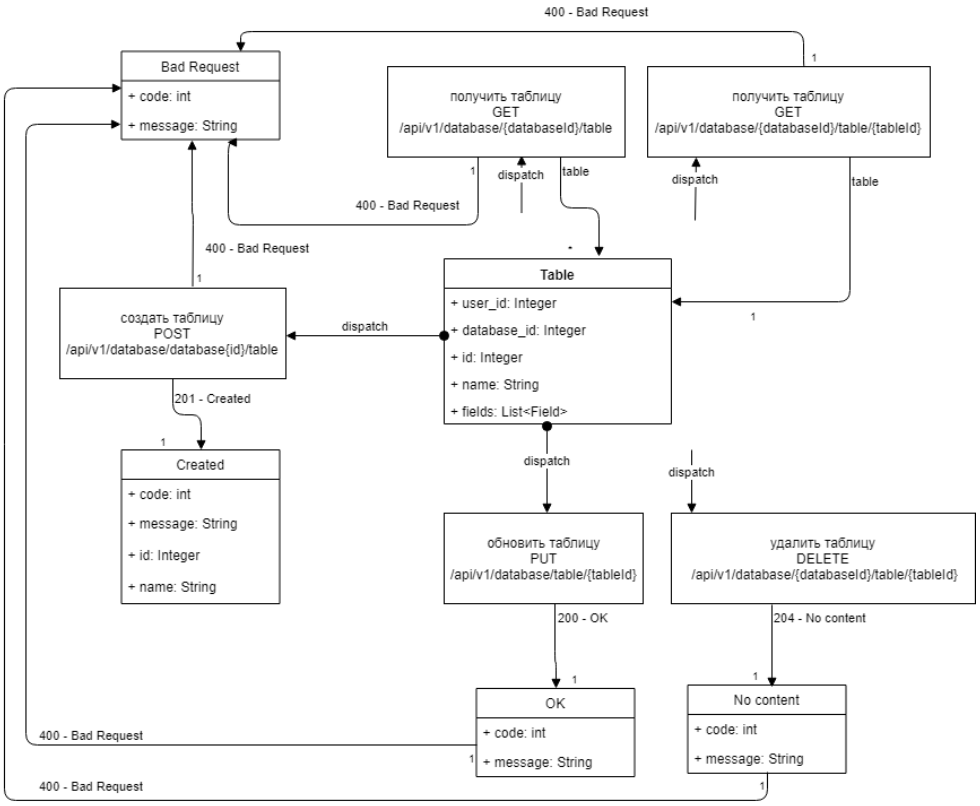

Рис. 31. Диаграмма HTTP запросов для таблиц

На диаграмме HTTP запросов для полей таблиц (Рис. 32) продемонстрировано, из чего состоит сущность Field: user_id – уникальный идентификатор пользователя, database_id – уникальный идентификатор базы данных, table_id – уникальный идентификатор таблицы, id – идентификатор поля, name – наименование поля, m_type – тип поля, default_value – значение поля по умолчанию, is_primary – признак первичного ключа, is_index – признак того, что поле является индексом, is_nullable – признак того, что поле может принимать значение NULL, is_unique – признак того, что поле должно быть уникальным, description – текстовое описание поля. Операции доступные для сущности «поле»:

- создание – POST запрос по URI – «/api/v1/database/{databaseId}/table/{tableId}/field», где databaseId – идентификатор базы данных, tableId – идентификатор таблицы;
- получение всех полей таблицы – GET запрос по URI – «/api/v1/database/{databaseId}/table/{tableId}/field»;

- получение поля по уникальным идентификаторам базы данных, таблицы и поля – GET запрос по URI – «/api/v1/database/{databaseId}/table/{tableId}/field/{fieldId}», где fieldId – идентификатор поля;
- обновление поля по уникальным идентификаторам базы данных, таблицы и поля – PUT запрос по URI – «/api/v1/database/{databaseId}/table/{tableId}/field/{fieldId}»;
- удаление поля по уникальным идентификаторам базы данных, таблицы и поля – DELETE запрос по URI – «/api/v1/database/{databaseId}/table/{tableId}/field/{fieldId}».

Возможные ответы сервера– неверный запрос (код 400 Bad Request), поле создано (код 201 Created), поле обновлено (код 200 OK), поле удалено (код 204 No content).

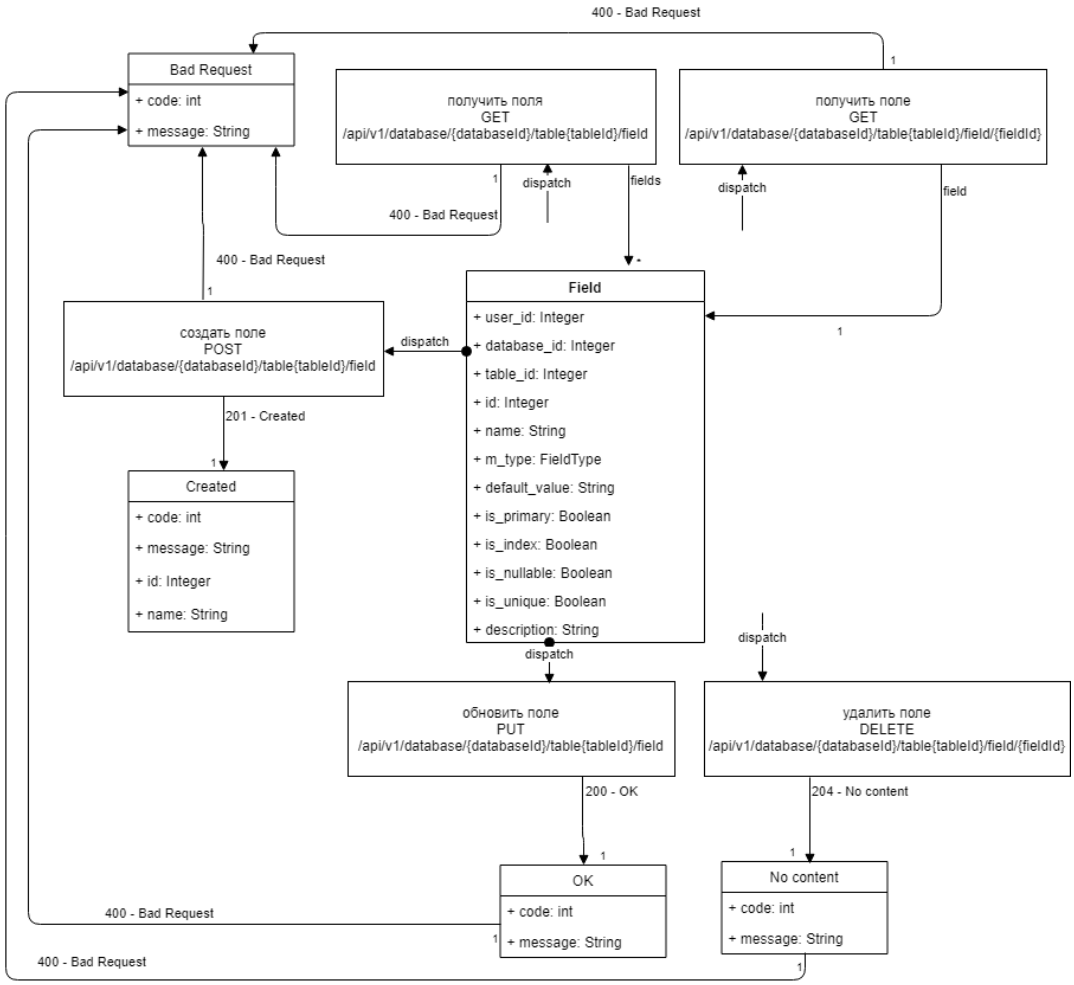

Рис. 32. Диаграмма HTTP запросов для полей

На последней диаграмме HTTP запросов для связей (Рис. 33) продемонстрировано, из чего состоит сущность Relationship: user_id – уникальный идентификатор пользователя, database_id – уникальный идентификатор базы данных, table1_id – идентификатор связываемой таблицы, field1_id – идентификатор связываемого поля, table2_id – идентификатор таблицы, с которой будет построена связь, field2_id – идентификатор поля, с которым будет построена связь, name – наименование связи, on_delete – тип действия, выполняемый при удалении связи, on_update – тип действия, выполняемый при обновлении связи, m_type – тип связи. Операции доступные для сущности «связь»:

- создание – POST запрос по URI – «/api/v1/database/{databaseId}/relationship», где databaseId – идентификатор базы данных;
- получение всех связей базы данных – GET запрос по URI – «/api/v1/database/{databaseId}/relationship»;
- получение связи по уникальным идентификаторам базы данных и связи – GET запрос по URI – «/api/v1/database/{databaseId}/relationship/{relationshipId}», где relationshipId – идентификатор связи;
- обновление связи по уникальным идентификаторам базы данных и связи – PUT запрос по URI – «/api/v1/database/{databaseId}/relationship/{relationshipId}»;
- удаление связи по уникальным идентификаторам базы данных и связи – DELETE запрос по URI – «/api/v1/database/{databaseId}/relationship/{relationshipId}».

Возможные ответы сервера – неверный запрос (код 400 Bad Request), связь создана (код 201 Created), связь обновлена (код 200 OK), связь удалена (код 204 No content).

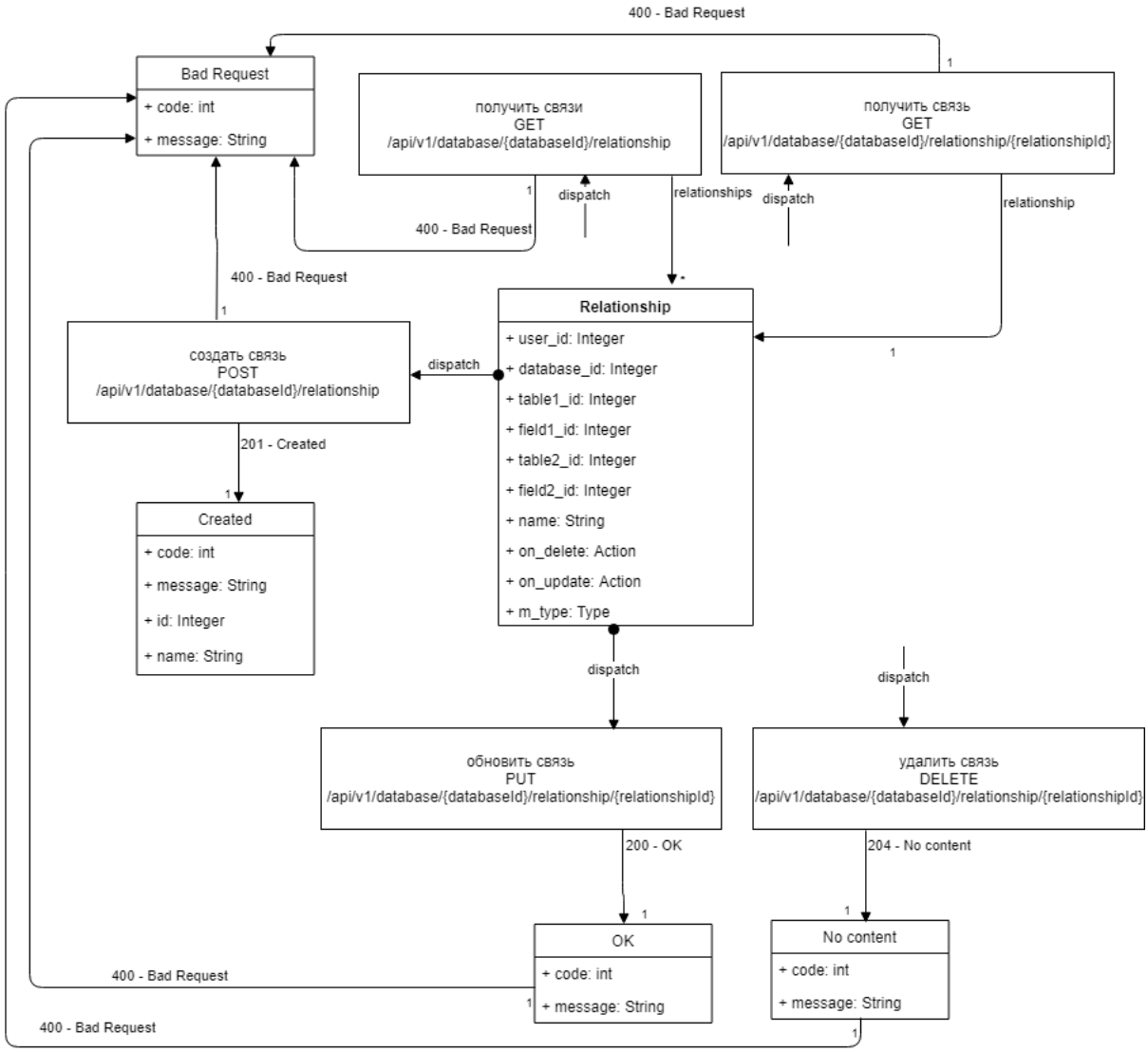

Рис. 33. Диаграмма HTTP запросов для связей

Далее следует рассмотреть диаграммы классов серверной и клиентской части приложения, чтобы чётко проследить зависимости между составными компонентами.

Рассмотрим диаграмму классов серверной части приложения (Рис. 34). На диаграмме можно выделить 5 основных сущностей – User, Database, Table, Field и Relationship. Также на диаграмме изображены 3 вспомогательные сущности, которые являются перечислениями (Enumeration) – FieldType, Type и Action. Рассмотрим представленные сущности более подробно.

- User – это базовая сущность, являющаяся моделью пользователя и содержит в себе его персональные данных, такие как пароль, имя пользователя и т.д.:

- Database – это базовая сущность, являющаяся моделью базы данных. Она содержит в себе информацию о наименовании базы данных, её таблицах и связях. Имеет 2 зависимости типа один ко многим по отношению к сущностям Table и Relationship;
- Table – это базовая сущность, являющаяся моделью таблицы базы данных и содержит в себе информацию о наименовании таблицы и принадлежащих ей полям. Имеет зависимость типа один ко многим по отношению к сущности Field и зависимость типа один к одному по отношению к сущности Relationship;
- Field – это базовая сущность, являющаяся моделью поля таблицы, она содержит в себе информацию о наименовании поля, типе данных, значения по умолчанию и пр. Имеет зависимость типа один к одному по отношению к перечислению FieldType (тип поля) и зависимость типа многие к одному относительно сущности Table;
- Relationship – это базовая сущность, являющаяся моделью связи между полями таблицы, она содержит информацию о наименовании связи, связываемых табличных полей, типах действия при удалении/обновлении и пр. Имеет зависимость типа один ко многим по отношению к сущности Table, зависимость типа многие к одному по отношению к сущности Database и 2 зависимости типа один к одному относительно перечислений Type и Action;
- FieldType – это сущность, которая является перечислением и содержит в себе обобщенные типы данных для баз данных;
- Action – это сущность, которая является перечислением и содержит в себе типы действий при удалении/обновлении связей в БД, такие как RESTRICT, NO_ACTION, CASCADE;
- Type – это сущность, которая является перечислением и содержит в себе типы связей между полями, такие как «один к одному» («one to one»), «один ко многим» («one to many»), «многие ко многим» («many to many»);

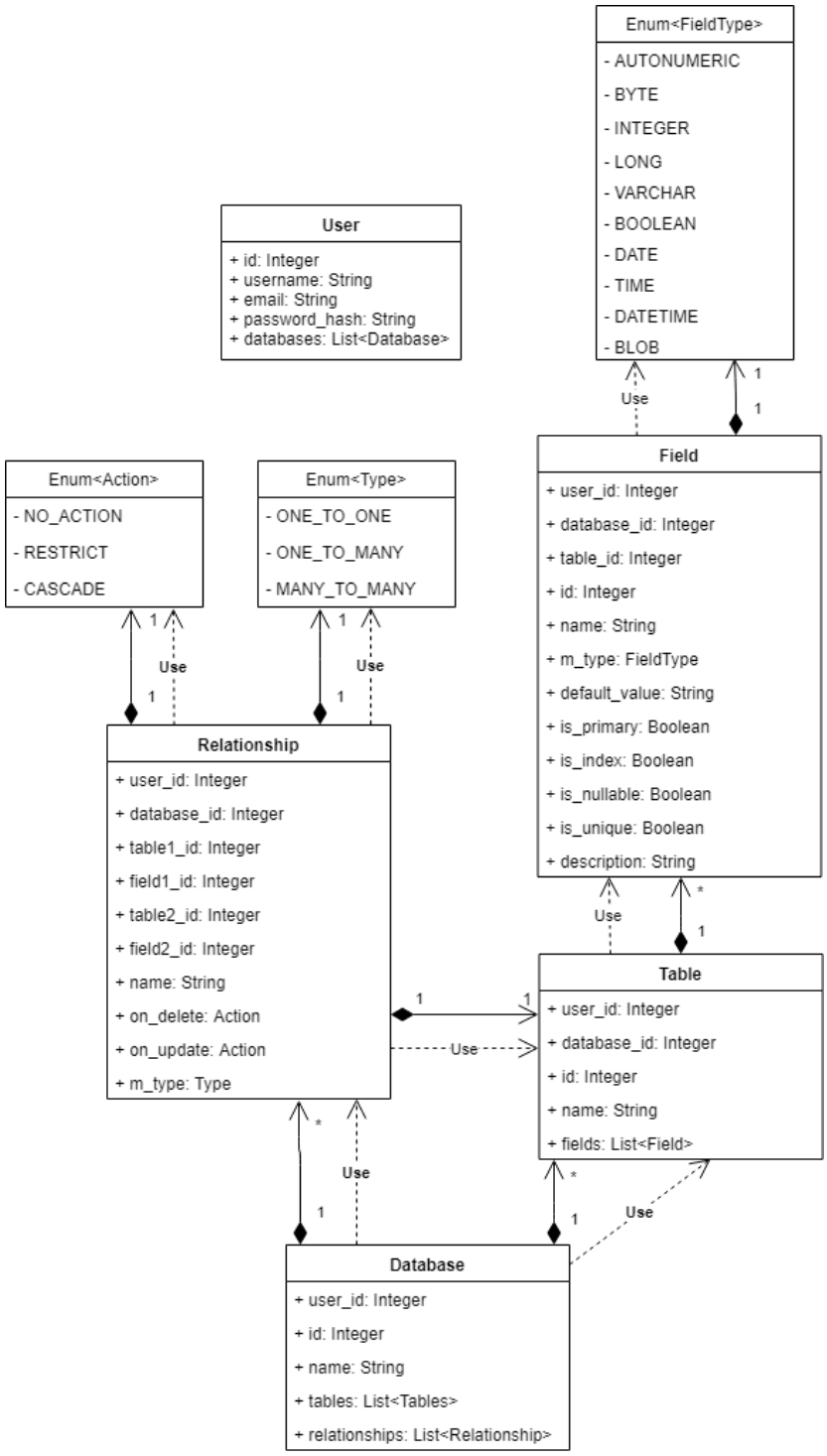

Рис. 34. Диаграмма доменных классов серверной части приложения

Рассмотрим также диаграммы классов клиентской части приложения. Представленные диаграммы поделены по внутренним пакетам приложения, которые были рассмотрены ранее.

На диаграмме классов пакета controllers (Рис. 35) проиллюстрированы интерфейсы, предоставляющее маппинг API сервера – UserApi, DatabaseApi,

TableApi, FieldApi, RelationshipApi. С помощью данные интерфейсов осуществляются HTTP запросы посредством класса Call, который принадлежит фреймворку Retrofit. Также все интерфейсы используют класс ControllerNames, который содержит в себе URI константы для маппинга.

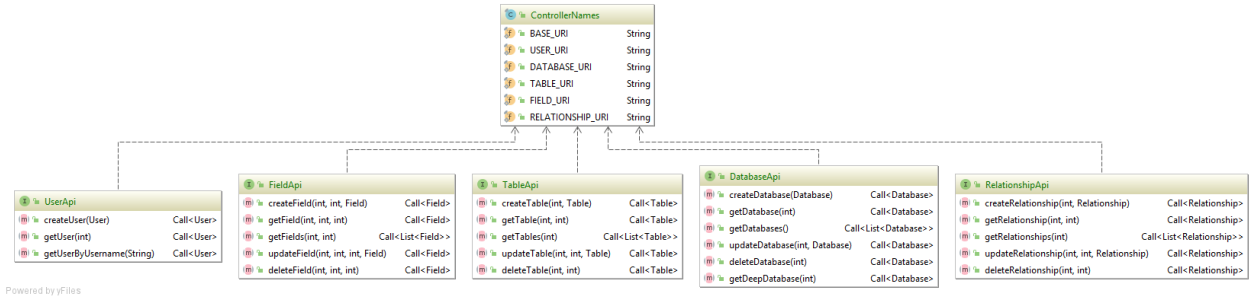

Рис. 35. Диаграмма классов пакета controllers

Диаграмма классов пакета domains (Рис. 36) почти полностью повторяет диаграмму классов серверной части приложения. Это необходимо для поддержки объектно-реляционного отображения (ORM) при HTTP запросах. Также, помимо повторных сущностей, в пакете domains содержаться другие классы, и интерфейсы. Например, класс ObjectState является перечислением (Enumeration) и представляет собой модель состояния объекта в двух режимах: объект является новым (NEW) и объект уже был создан (CREATED), а интерфейс DatabaseRelatedObject необходим для обобщения сущностей, которые относятся к базе данных (Database, Table, Field, Relationship).

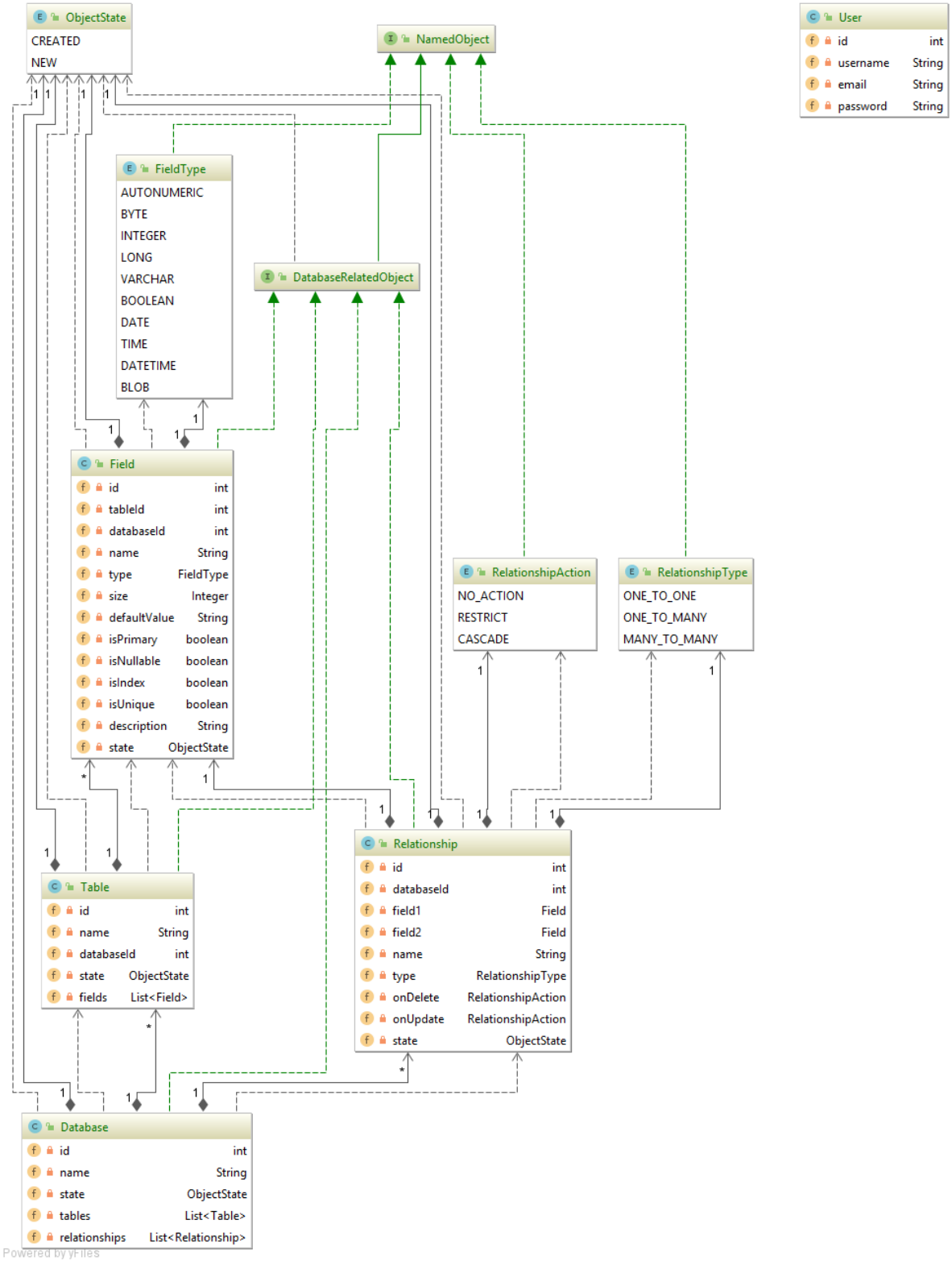

Рис. 36. Диаграмма классов пакета domains

Диаграмма классов пакета generators (Рис. 37) содержит в себе 4 класса.

- DDLGenerator – это абстрактный класс, который содержит в себе базовую реализацию генерации SQL кода;

- PostrgreSQLDDLGenerator и SQLiteDDLGenerator – потомки класса DDLGenerator и содержат в себе реализацию генерации SQL кода для разных диалектов (PostgreSQL и SQLite);
- DatabaseType – это перечисление (Enumeration) которое содержит данные о поддерживаемых SQL диалектов.

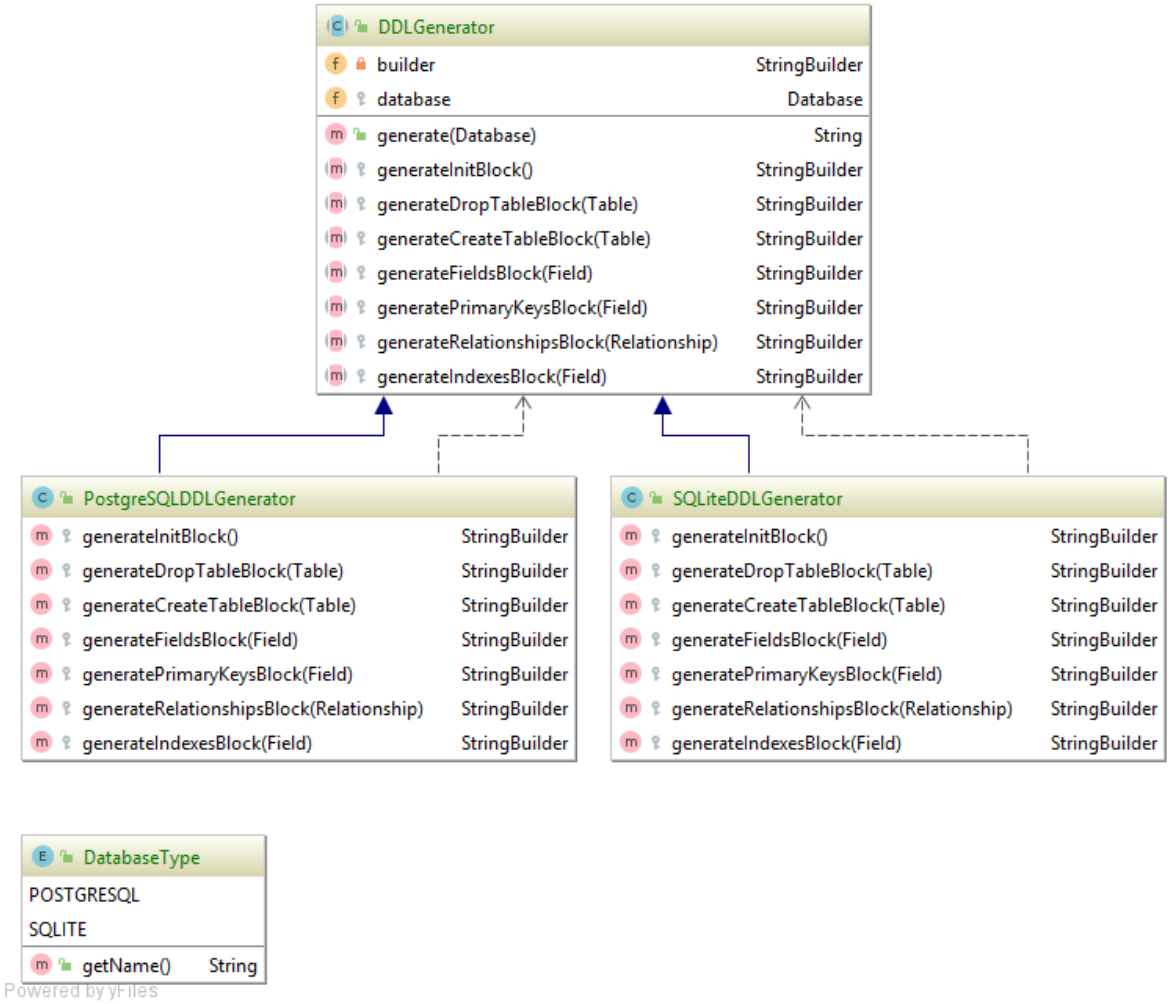

Рис. 37. Диаграмма классов пакета generators

Рассмотрим диаграмму классов пакета services (Рис. 38). Интерфейс Base-Service задает поведение для сервисов, предназначенных для CRUD операций над основными сущностями базы данных (DatabaseService, TableService, Field-Service, RelationshipService), которые в свою очередь продуцируют реализацию HTTP запросов, обобщенный код которой находится в наследуемом классе

CallbackService. Класс UiService является фабрикой пользовательских View компонентов, а класс ServiceGenerator является фабрикой для создания API интерфейсов (DatabaseApi, TableApi, FieldApi, RelationshipApi).

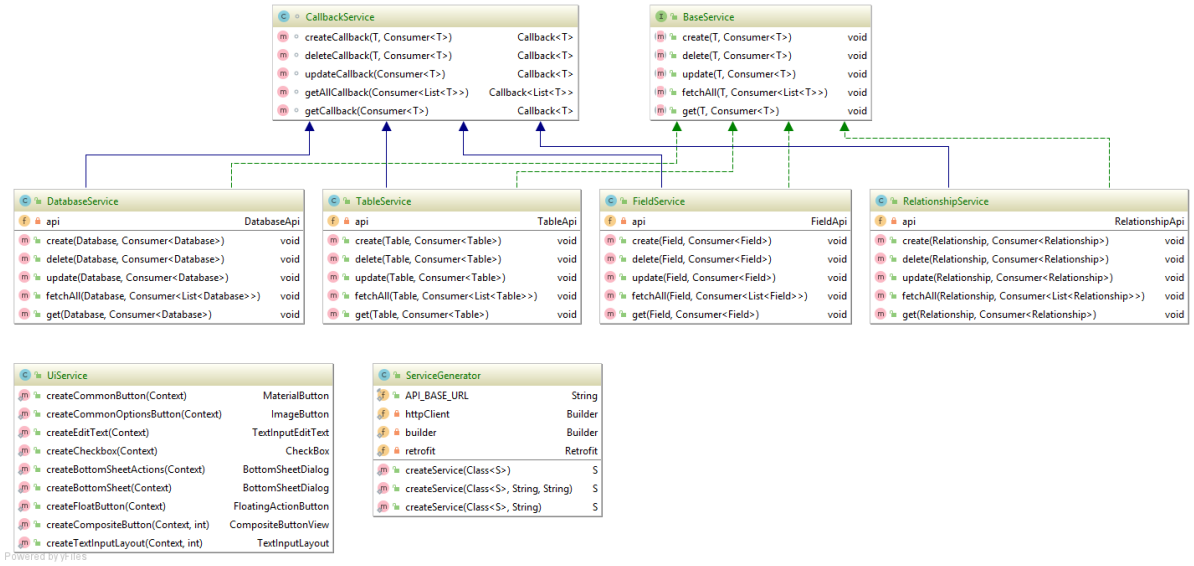

Рис. 38. Диаграмма классов пакета services

Рассмотрим классы, представленные на диаграмме классов пакета ui (Рис. 39). В этом пакете в основном содержаться классы Activity. Все Activity, на которых происходит взаимодействие с базовыми сущностями базы данных обобщены по интерфейсу DatabaseRelatedObject, поэтому логика поведения интерфейса по умолчанию находится в классе BaseActiviy, которая инкапсулирует и переопределяет основные взаимодействия в дочерние классы (DatabaseActivity, TableActivity, FieldActivity, RelationshipActivity). Класс NavigationActivity стоит на самом верху иерархии среди Activity, это необходимо для того чтобы все Activity имели боковую панель (NavigationDrawer), также почти все Activity реализуют интерфейс Loadable, предназначенный для того чтобы система переходила в состояние ожидания во время выполнения асинхронного HTTP запроса к серверу. При запуске приложения открывается стартовая Activity – MainEmptyActivity, которая визуально является пустой, это необходимо для того чтобы определить авторизован ли пользователь. Если пользователь авторизован выполняется переход на главную Activity приложения (DatabaseActivity), иначе выполняется переход на Activity с входом в систему (LoginActivity).

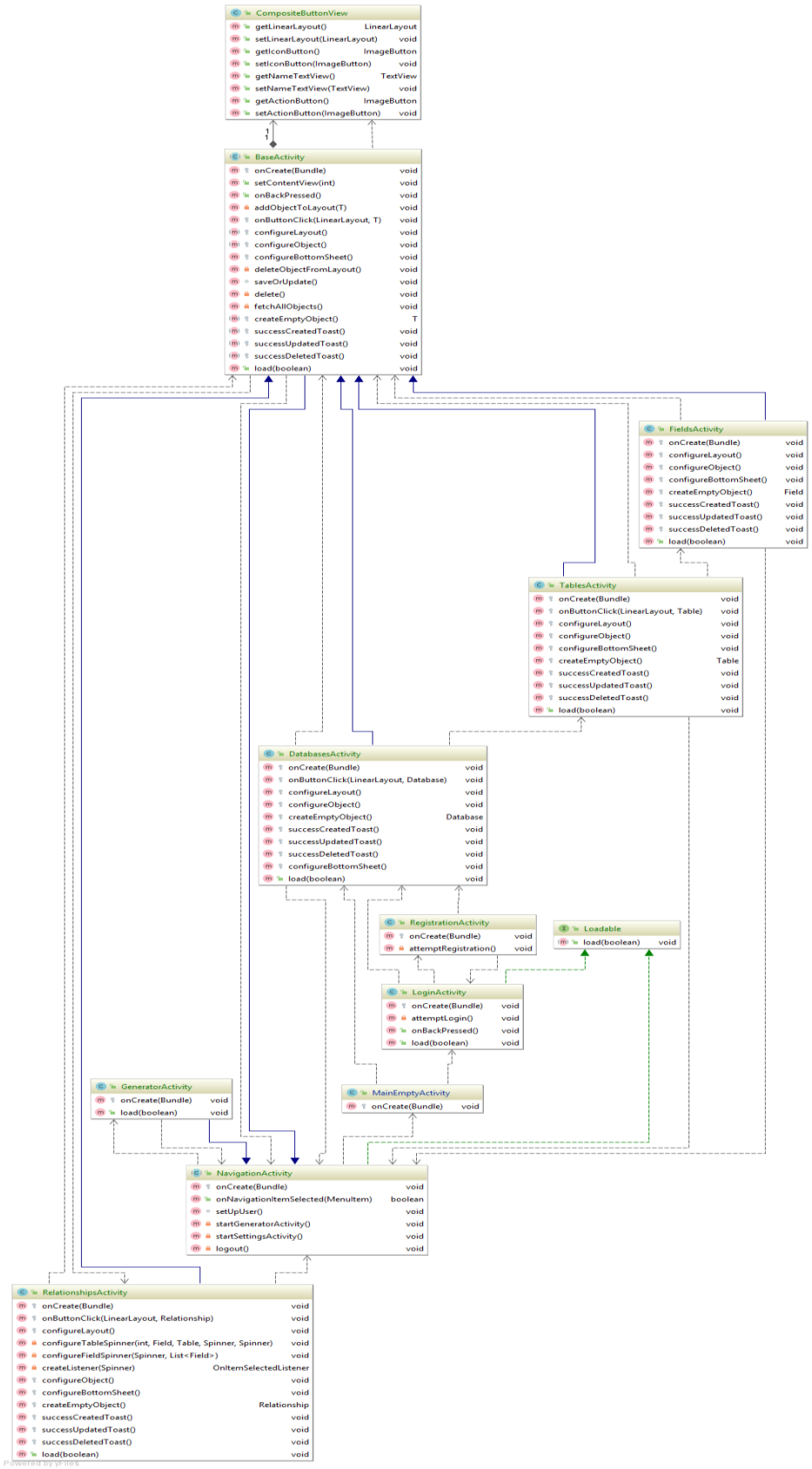

Рис. 39. Диаграмма классов пакета ui

На последней диаграмме классов продемонстрированы классы пакета utils (Рис. 40). Этот пакет содержит в себе всего 3 утилиты.

- NamedObjectAdapter – это класс принимающий в качестве параметра обобщение по интерфейсу DatabaseRelaredObject. Данный класс служит для полиморфного поведения по умолчанию пользовательского элемента интерфейса – Spinner;
- Validator – это класс для валидации логина, пароля, электронной почты при входе или регистрации пользователя;
- AuthenticationInterceptor – это класс позволяющих авторизовываться на сервере при помощи токена.

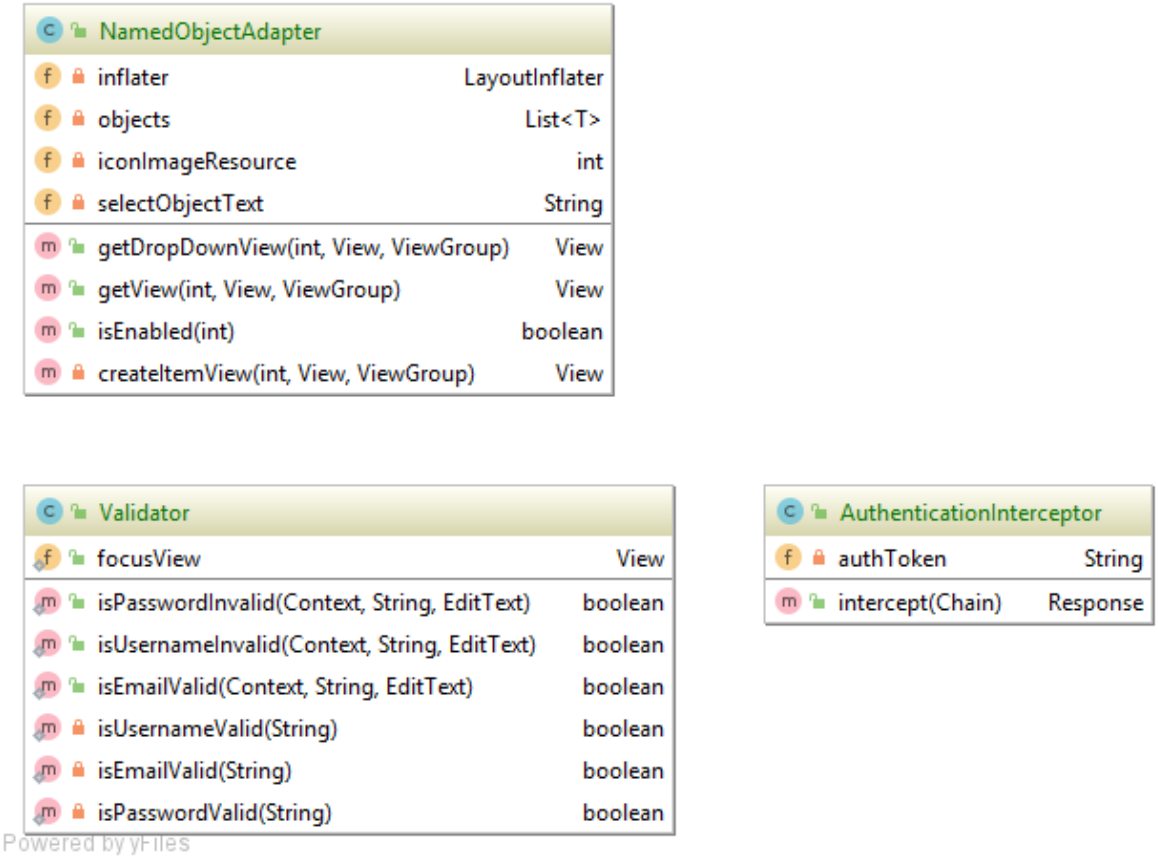

Рис. 40. Диаграмма классов пакета utils

Таким образом построенные диаграммы позволяют четко разграничить бизнес логику приложения, отразить механизмы взаимодействия всех частей комплексной системы между собой.

2.2 Описание приложения «EasyBox»

В качестве демонстрация функционирования приложения будут рассмотрены некоторые части исходного кода, уникальные алгоритмы, сторонние библиотеки.

Рассмотрим некоторые доменные классы приложения из пакета «domains». Интерфейс «DatabaseRelatedObject» (Листинг 1) задает поведение для классов с целью того, чтобы указать что классы, реализующие этот интерфейс, являются зависимыми от базы данных объектами.

```
package ru.sanaxes.easybox.domains;

/**
 * Интерфейс зависимого от БД объекта
 */
public interface DatabaseRelatedObject extends
NamedObject {
    int getId();
    void setId(int id);
    int getDatabaseId();
    void setDatabaseId(int databaseId);
    ObjectState getState();
    void setState(ObjectState state);
}
```

Листинг 1. Интерфейс DatabaseRelatedObject

DatabaseRelatedObject гарантирует что зависимые от базы данных классы должны иметь уникальный идентификатор, идентификатор базы данных, к которой принадлежит объект этого класса и наличие состояния класс ObjectState (Листинг 2).

```
package ru.sanaxes.easybox.domains;

/**
 * Состояние объекта
 */
public enum ObjectState {
    /**
     * Созданный объект
     */
    CREATED,
    /**
     * Новый объект
     */
    NEW
}
```

Листинг 2. Enum класс ObjectState

Классы «Database», «Table», «Field» и «Relationship» реализуют интерфейс «DatabaseRelatedObject» так как обладают всеми свойствами объекта, принадлежащего базе данных. Интерфейс «DatabaseRelatedObject» является очень важным связующим классом, так как по нему обобщены многие классы.

Рассмотрим на примере такие обобщения. Базовый абстрактный класс Activity для выполнения CRUD операций над объектами, зависимыми от базы данных, называется BaseActivity (Листинг 3). При наследовании от данного класса необходимо явно указать с каким из классов, реализующих интерфейс «DatabaseRelatedObject», будет происходить взаимодействие.

```
/**
 * Базовый класс-представление Activity для работы с DatabaseRelatedObject
объектами
 *
 * @param <T> - DatabaseRelatedObject объект
 */
public abstract class BaseActivity<T extends DatabaseRelatedObject> extends
NavigationActivity {
…
}
```

Листинг 3. Класс BaseActivity

Класс Activity баз данных (Листинг 4) наследуется от класса BaseActivity и в качестве параметра передает класс Database. Обобщение классов позволило значительно сократить объем исходного кода за счёт одинаковых операций. Классы Activity, наследуемые от DatabaseRelatedObject типовые: в качестве полей указывается класс с Api текущего класса и View объекты для редактирования зависимых объектов. Некоторые методы, принадлежащие классу BaseActivity переопределяются, например onButtonClick на каждой Activity отвечает за действие, которое произойдет при нажатии на макет (переход на следующую Activity) и для каждой Activity действия разные: из DatabaseActivity переход в TablesActivity (Листинг 5) далее в FieldActivity (Листинг 6).

```
package ru.sanaxes.easybox.ui;

import android.content.Intent;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.widget.LinearLayout;
import com.google.android.material.textfield.TextInputLayout;
import ru.sanaxes.easybox.R;
import ru.sanaxes.easybox.controllers.DatabaseApi;
import ru.sanaxes.easybox.domains.Database;
import ru.sanaxes.easybox.services.DatabaseService;
import ru.sanaxes.easybox.services.ServiceGenerator;
import ru.sanaxes.easybox.services.UiService;

/**
 * Activity Баз данных
 */
public class DatabasesActivity extends BaseActivity<Database> {

    // UI
    private TextInputLayout databaseNameInputLayout;
    // Logic
    private DatabaseApi api;

    @Override
    protected void onCreate(Bundle savedInstanceState) {    }

    @Override
    protected void onButtonClick(LinearLayout linearLayout, Database data-
base) {    }

    @Override
    protected void configureLayout() {    }

    @Override
    protected void configureObject() {    }

    @Override
    protected Database createEmptyObject() {    }

    @Override
    protected void successCreatedToast() {    }

    @Override
    protected void successUpdatedToast() {    }

    @Override
    protected void successDeletedToast() {    }

    @Override
    protected void configureBottomSheet() {    }

    @Override
    public void load(boolean enabled) {    }
}
```

Листинг 4. Класс DatabasesActivity

```java
package ru.sanaxes.easybox.ui;

import android.content.Intent;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.widget.LinearLayout;
import com.google.android.material.textfield.TextInputLayout;
import ru.sanaxes.easybox.R;
import ru.sanaxes.easybox.controllers.TableApi;
import ru.sanaxes.easybox.domains.Table;
import ru.sanaxes.easybox.services.ServiceGenerator;
import ru.sanaxes.easybox.services.TableService;
import ru.sanaxes.easybox.services.UiService;

/**
 * Activity Таблиц
 */
public class TablesActivity extends BaseActivity<Table> {

    // UI
    private TextInputLayout tableNameInputLayout;
    // Logic
    private DatabaseApi api;

    @Override
    protected void onCreate(Bundle savedInstanceState) {    }

    @Override
    protected void onButtonClick(LinearLayout linearLayout, Table table) {
}

    @Override
    protected void configureLayout() {    }

    @Override
    protected void configureObject() {    }

    @Override
    protected Table createEmptyObject() {    }

    @Override
    protected void successCreatedToast() {    }

    @Override
    protected void successUpdatedToast() {    }
}
```

Листинг 5. Класс Activity таблиц

```
package ru.sanaxes.easybox.ui;

import …

/**
 * Activity полей
 */
public class FieldsActivity extends BaseActivity<Field> {

    private TextInputLayout nameInputLayout;
    private Spinner typeSpinner;
    private NamedObjectAdapter<FieldType> fieldTypeAdapter;
    private TextInputLayout sizeInputLayout;
    private TextInputLayout defaultValueInputLayout;
    private CheckBox isPrimaryCheckBox;
    private CheckBox isNullableCheckBox;
    private CheckBox isIndexCheckBox;
    private CheckBox isUniqueCheckBox;
    private TextInputLayout descriptionInputLayout;

    private FieldApi api;

    @Override
    protected void onCreate(Bundle savedInstanceState) {    }

    @Override
    protected void configureLayout() {    }

    @Override
    protected void configureObject() {    }

    @Override
    protected void configureBottomSheet {    }

    @Override
    protected Field createEmptyObject() {    }

    @Override
    protected void successCreatedToast() {    }

    @Override
    protected void successUpdatedToast() {    }

    @Override
    protected void successDeletedToast() {    }

    @Override
    public void load(boolean enabled) {    }

}
```

Листинг 6. Класс Activity полей

```
package ru.sanaxes.easybox.ui;

import …

public class RelationshipsActivity extends BaseActivity<Relationship> {
    private TextInputLayout nameInputLayout;
    private Spinner table1Spinner;
    private Spinner field1Spinner;
    private Spinner table2Spinner;
    private Spinner field2Spinner;
    private Spinner typeSpinner;
    private Spinner onDeleteSpinner;
    private Spinner onUpdateSpinner;
    private NamedObjectAdapter<Table> table1Adapter;
    private NamedObjectAdapter<Table> table2Adapter;
    private NamedObjectAdapter<RelationshipAction> onDeleteAdapter;
    private NamedObjectAdapter<RelationshipAction> onUpdateAdapter;
    private NamedObjectAdapter<RelationshipType> typeAdapter;

    private RelationshipApi api;
    private static int countOfThreads = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {    }

    @Override
    protected void configureLayout()    }

    private void configureFieldAdapter(Field field, NamedObjectAdapter<Table>
tableAdapter, Spinner tableSpinner, Spinner fieldSpinner) {    }

    private void configureFieldSpinner(Spinner fieldSpinner, List<Field>
fields) {    }

    private AdapterView.OnItemSelectedListener createListener(Spinner field-
Spinner) {    }

    @Override
    protected void configureObject() {    }

    @Override
    protected void configureBottomSheet {    }

    @Override
    protected Relationship createEmptyObject() {    }

    @Override
    protected void successCreatedToast() {    }
}
```

Листинг 7. Activity связей

Интерфейс BaseService (Листинг 8) также является обобщенным по интерфейсу DatabaseRelatedObject. Классы DatabaseService, TableService, TableService и RelationshipService реализуют интерфейс BaseService, который задает поведение для CRUD операций над классами, зависимыми от базы данных.

```java
package ru.sanaxes.easybox.services;

import ru.sanaxes.easybox.domains.DatabaseRelatedObject;

import java.util.List;
import java.util.function.Consumer;

/**
 * Интерфейс для CRUD операций
 *
 * @param <T> - Именованный объект
 */
public interface BaseService<T extends DatabaseRelatedObject> {
    /**
     * Создание объекта
     *
     * @param object - объект
     * @param action - действие которое необходимо выполнить после успешного
создания объекта
     */
    void create(T object, Consumer<T> action);

    /**
     * Удаление объекта
     *
     * @param object - объект
     * @param action - действие которое необходимо выполнить после успешного
удаления объекта
     */
    void delete(T object, Consumer<T> action);

    /**
     * Обновление обхекта
     *
     * @param object - объект
     * @param action - действие которое необходимо выполнить после успешного
обновления объекта
     */
    void update(T object, Consumer<T> action);

    /**
     * Поулчение всех объектов
     *
     * @param object - объект
     * @param action - действие которое необходимо выполнить после успешного
полуения объектов*/
```

Листинг 8. Интерфейс BaseService

Класс CallbackService (Листинг 9) – это еще один класс, обобщенный по интерфейсу DatabaseRelatedObject. Это абстрактный класс, содержит непосредственно реализацию CRUD операций в виде HTTP запросов с помощью фреймворка Retrofit.

```
package ru.sanaxes.easybox.services;

import androidx.annotation.NonNull;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import ru.sanaxes.easybox.domains.DatabaseRelatedObject;
import ru.sanaxes.easybox.domains.ObjectState;

import java.util.List;
import java.util.function.Consumer;

/**
 * Сервис для выполнения CRUD запросов
 *
 * @param <T> - объект зависимый от БД
 */
class CallbackService<T extends DatabaseRelatedObject> {

    Callback<T> createCallback(T object, Consumer<T> action) {
        return new Callback<T>() {
            @Override
            public void onResponse(Call<T> call, Response<T> response) {
                T created = response.body();
                if (response.isSuccessful() && created != null) {
                    object.setId(created.getId());
                    object.setState(ObjectState.CREATED);
                    action.accept(object);
                }
            }

            @Override
            public void onFailure(Call<T> call, Throwable t) {

            }
        };
    }

    Callback<T> deleteCallback(T object, Consumer<T> action) {    }

    Callback<T> updateCallback(Consumer<T> action) {    }

    Callback<List<T>> getAllCallback(Consumer<List<T>> action) {    }

    Callback<T> getCallback(Consumer<T> action) {    }

}
```

Листинг 9. Класс CallbackService

Классы DatabaseService (Листинг 10), TableService (Листинг 11), FieldService (Листинг 12) и RelationshipService (Листинг 13) реализуют интерфейс BaseService и наследуются от класса CallbackService, что позволяет сделать реализацию CRUD операций одинаковой для всех сущностей, отличаться будет только URI для HTTP запроса.

```
package ru.sanaxes.easybox.services;

import ru.sanaxes.easybox.controllers.DatabaseApi;
import ru.sanaxes.easybox.domains.Database;

import java.util.List;
import java.util.function.Consumer;

/**
 * CRUD Сервис для БД
 */
public class DatabaseService extends CallbackService<Database> implements
BaseService<Database> {

    private DatabaseApi api;

    public DatabaseService(DatabaseApi api) {
        this.api = api;
    }

    @Override
    public void create(Database object, Consumer<Database> onSuccessCreate) {
        api.createDatabase(object).enqueue(super.createCallback(object, on-
SuccessCreate));
    }

    @Override
    public void delete(Database object, Consumer<Database> action) {
        api.deleteDatabase(object.getId()).enqueue(super.deleteCallback(ob-
ject, action));
    }

    @Override
    public void update(Database object, Consumer<Database> action) {
        api.updateDatabase(object.getId(), object).enqueue(super.update-
Callback(action));
    }

    @Override
    public void fetchAll(Database dummy, Consumer<List<Database>> action) {
        api.getDatabases().enqueue(super.getAllCallback(action));
    }

    @Override
    public void get(Database database, Consumer<Database> action) {
        api.getDeepDatabase(database.getId()).enqueue(super.getCallback(ac-
tion));
    }
}
```

Листинг 10. Класс DatabaseService

```
package ru.sanaxes.easybox.services;

import ru.sanaxes.easybox.controllers.TableApi;
import ru.sanaxes.easybox.domains.Table;

import java.util.List;
import java.util.function.Consumer;

public class TableService extends CallbackService<Table> implements BaseServ-
ice<Table> {

    private TableApi api;

    public TableService(TableApi api) {
        this.api = api;
    }

    @Override
    public void create(Table object, Consumer<Table> action) {
        api.createTable(object.getDatabaseId(), object).enqueue(super.create-
Callback(object, action));
    }

    @Override
    public void delete(Table object, Consumer<Table> action) {
        api.deleteTable(object.getDatabaseId(), object.getId()).enqueue(su-
per.deleteCallback(object, action));
    }

    @Override
    public void update(Table object, Consumer<Table> action) {
        api.updateTable(object.getDatabaseId(), object.getId(), ob-
ject).enqueue(super.updateCallback(action));
    }

    @Override
    public void fetchAll(Table table, Consumer<List<Table>> action) {
        api.getTables(table.getDatabaseId()).enqueue(super.getAllCallback(ac-
tion));
    }

    @Override
    public void get(Table object, Consumer<Table> action) {
        api.getTable(object.getDatabaseId(), object.getId()).enqueue(su-
per.getCallback(action));
    }


}
```

Листинг 11. Класс TableService

```
package ru.sanaxes.easybox.services;

import ru.sanaxes.easybox.controllers.FieldApi;
import ru.sanaxes.easybox.domains.Field;

import java.util.List;
import java.util.function.Consumer;

public class FieldService extends CallbackService<Field> implements BaseServ-
ice<Field> {

    private FieldApi api;

    public FieldService(FieldApi api) {
        this.api = api;
    }

    @Override
    public void create(Field object, Consumer<Field> action) {
        api.createField(object.getDatabaseId(), object.getTableId(), ob-
ject).enqueue(super.createCallback(object, action));
    }

    @Override
    public void delete(Field object, Consumer<Field> action) {
        api.deleteField(object.getDatabaseId(), object.getTableId(), ob-
ject.getId()).enqueue(super.deleteCallback(object, action));
    }

    @Override
    public void update(Field object, Consumer<Field> action) {
        api.updateField(object.getDatabaseId(), object.getTableId(), ob-
ject.getId(), object).enqueue(super.updateCallback(action));
    }

    @Override
    public void fetchAll(Field field, Consumer<List<Field>> action) {
        api.getFields(field.getDatabaseId(), field.getTableId()).enqueue(su-
per.getAllCallback(action));
    }

    @Override
    public void get(Field object, Consumer<Field> action) {
        api.getField(object.getDatabaseId(), object.getTableId(), ob-
ject.getId()).enqueue(super.getCallback(action));
    }

}
```

Листинг 12. Класс FieldService

```
package ru.sanaxes.easybox.services;

import ru.sanaxes.easybox.controllers.RelationshipApi;
import ru.sanaxes.easybox.domains.Relationship;

import java.util.List;
import java.util.function.Consumer;

public class RelationshipService extends CallbackService<Relationship> imple-
ments BaseService<Relationship> {

    private RelationshipApi api;

    public RelationshipService(RelationshipApi api) {
        this.api = api;
    }

    @Override
    public void create(Relationship object, Consumer<Relationship> action) {
        api.createRelationship(object.getDatabaseId(), object).enqueue(su-
per.createCallback(object, action));
    }

    @Override
    public void delete(Relationship object, Consumer<Relationship> action) {
        api.deleteRelationship(object.getDatabaseId(), ob-
ject.getId()).enqueue(super.deleteCallback(object, action));
    }

    @Override
    public void update(Relationship object, Consumer<Relationship> action) {
        api.updateRelationship(object.getDatabaseId(), object.getId(), ob-
ject).enqueue(super.updateCallback(action));
    }

    @Override
    public void fetchAll(Relationship relationship, Consumer<List<Relation-
ship>> action) {
        api.getRelationships(relationship.getDatabaseId()).enqueue(super.get-
AllCallback(action));
    }

    @Override
    public void get(Relationship object, Consumer<Relationship> action) {
        api.getRelationship(object.getDatabaseId(), ob-
ject.getId()).enqueue(super.getCallback(action));
    }


}
```

Листинг 13. Класс RelationshipService

Рассмотрим программную реализацию классов для генерации SQL кода из пакета generators. Класс DatabaseType (Листинг 14) представляет собой enum класс со списком поддерживаемых целевых СУБД для генерации кода. На данный момент поддерживаются такие СУБД как PostgreSQL и SQLite.

```java
package ru.sanaxes.easybox.generators;

import ru.sanaxes.easybox.domains.NamedObject;

/**
 * Целевая СУБД
 */
public enum DatabaseType implements NamedObject {
    POSTGRESQL,
    SQLITE;

    @Override
    public String getName() {
        return super.name();
    }
}
```

Листинг 14. Enum класс DatabaseType

Класс SQLGenerator (Листинг 15) является абстрактным и содержит в себе: поле builder типа StringBuilder для удобной конкатенации строк, поле объект базы данных Database database, метод generate – генерация всего SQL кода, абстрактные методы generateInitBlock генерация кода инициализации БД, generateDropTableBlock генерация кода удаления таблиц, generateCreateTableBlock генерация кода создания таблиц, generateFieldsBlock генерация кода создания полей, generatePrimaryKeysBlock генерация кода создания первичных ключей, generateRelationshipsBlock генерация кода создания отношений, generateIndexesBlock генерация кода создания индексов. Алгоритм генерации кода всего кода (Листинг 16) в общем виде устроен следующим образом: составляется список пар типа «таблица-поле этой таблицы, имеющее связь», далее генерируется блок создания базы данных. Составленный список поочередно итерируется и на каждой итерации генерируется блок каждой таблицы, её поля, связи, первичные ключи и связи. В конце алгоритма происходит преобразование накопленного кода в классе StringBuilder в одну строку.

```
package ru.sanaxes.easybox.generators;

import androidx.core.util.Pair;
import ru.sanaxes.easybox.domains.Database;
import ru.sanaxes.easybox.domains.Field;
import ru.sanaxes.easybox.domains.Relationship;
import ru.sanaxes.easybox.domains.Table;

import java.util.List;
import java.util.stream.Collectors;

public abstract class SQLGenerator {

    private StringBuilder builder = new StringBuilder();

    protected Database database;

    public String generate(Database database) {   }

    protected abstract StringBuilder generateInitBlock();
    protected abstract StringBuilder generateDropTableBlock(Table table);
    protected abstract StringBuilder generateCreateTableBlock(Table table);
    protected abstract StringBuilder generateFieldsBlock(Field field);
    protected abstract StringBuilder generatePrimaryKeysBlock(Field field);
    protected abstract StringBuilder generateRelationshipsBlock(Relationship
relationship);
    protected abstract StringBuilder generateIndexesBlock(Field field);
}
```

Листинг 15. Класс SQLGenerator

```
public String generate(Database database) {
    this.database = database;
    List<Pair<Table, List<Relationship>>> list = database.getTables()
            .stream()
            .map(table -> new Pair<>(table, database.getRelationships()
                    .stream()
                    .filter(rel -> rel.getField2().getTableId() == ta-
ble.getId())
                    .collect(Collectors.toList())))
            .collect(Collectors.toList());
    builder.append(generateInitBlock());
    list.forEach(pair -> {
        Table table = pair.first;
        builder
                .append(generateDropTableBlock(table))
                .append(generateCreateTableBlock(table));
        List<Field> fields = table.getFields();
        fields.forEach(field -> builder.append(generateFieldsBlock(field)));
        fields.stream().filter(Field::isPrimary)
                .forEach(field -> builder.append(generatePrima-
ryKeysBlock(field)));
        builder.replace(builder.length() - 2, builder.length(), "");
        List<Relationship> relationships = pair.second;
        relationships.forEach(relationship -> builder.append(gener-
ateRelationshipsBlock(relationship)));
        builder
                .append("\n")
                .append(")")
                .append(";")
                .append("\n")
                .append("\n");
        fields.stream().filter(f -> f.isIndex() || f.isUnique())
                .forEach(field -> builder.append(generateIndex-
esBlock(field)));
    });
    return builder.toString();
}
```

Листинг 16. Алгоритм генерации SQL кода

Таким образом разработанное приложение имеет понятную и компактную структуру программного кода, построенную на основе парадигмы объектно-ориентированного программирования, с соблюдением её основных принципов: инкапсуляции, полиморфизма, наследования и абстракции.

2.3 Результаты апробации

Материалы работы прошли апробацию в формате публикации в сборнике студенческих научных работ[40]. Также апробация была проведана в формате экспертное на предприятии ООО «АльтероПауэр».

Экспертное оценивании проводилось на основе следующих критериев:

- удобство интерфейса;
- синхронизация на разных устройствах;
- функционал приложения;
- проектирование БД;
- качество выходных данных.

Экспертное оценивание проводилось среди респондентов, занимающих такие должности как системный администратор, старший программист, бизнес аналитик, разработчик баз данных. Результаты оценивания. По результатам апробации (Рис. 41, Рис. 42) можно сделать вывод о том, что разработанное приложение полностью удовлетворяет техническому заданию.

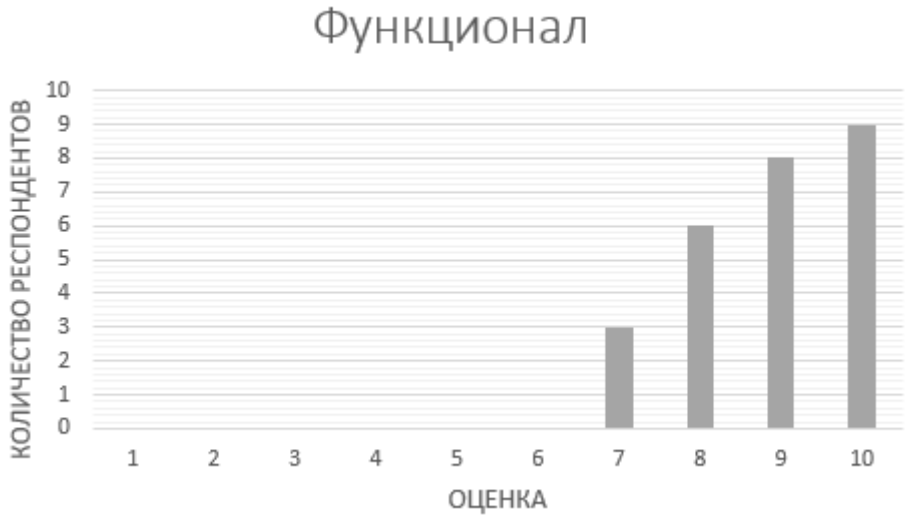

Рис. 41. Диаграмма апробации функционала приложения

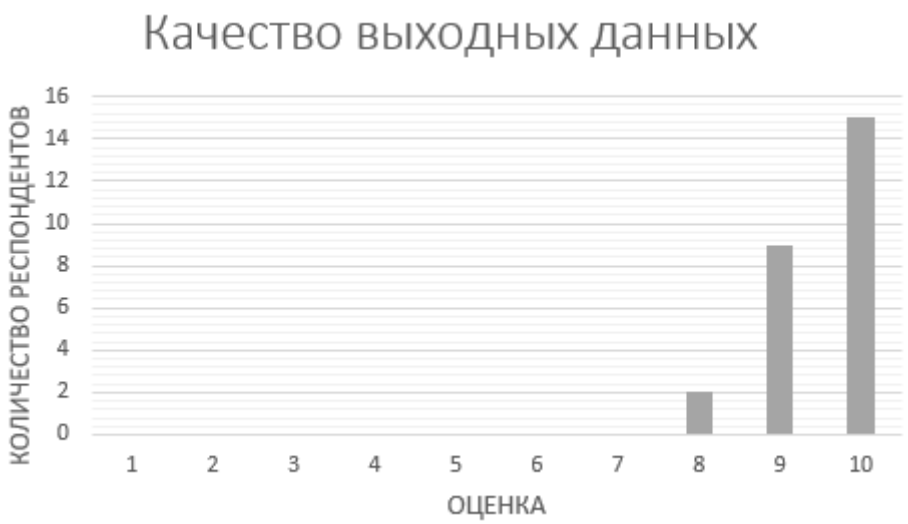

Рис. 42. Диаграмма апробации качества выходных данных приложения

Заключение

В ходе выпускной квалификационной работы было разработано мобильное приложение для проектирования баз данных и генерации SQL кода на устройства под управлением операционной системы Android «EasyBox».

Был проведен сравнительный анализ языков программирования и интегрированных сред разработки, позволяющих создавать мобильные приложения для операционной системы Android.

Построены информационные, функциональные и UML диаграммы, отражающие принципы работы, логику взаимодействия элементов приложения и пр.

Разработано приложение в соответствии с техническим заданием.

Также была проведена апробация, в формате экспертного оценивания и в формате публикации в сборнике студенческих научных работ.

Список информационных источников

1. 68ГОСТ Р 7.0.83-2013 «Система стандартов по информации, библиотечному и издательскому делу. Электронные издания. Основные виды и выходные сведения» от 15.10.2013 № 1163-ст // Стандартинформ. 2014 г. с изм. и допол. в ред. от 12.09.2018.
2. ГОСТ 2.105-95. Межгосударственный стандарт «Единая система конструкторской документации. Общие требования к текстовым документам» от 08.08.1995 № 426 // Всероссийским научно-исследовательским институтом стандартизации и сертификации в машиностроении (ВНИИНМАШ) Госстандарта России. 1995 г.
3. ГОСТ 19.502-78 ЕСПД. Описание применения. Требования к содержанию и оформлению (с Изменением N 1) от 01.01.1980 № 3350 // январь 2010 г. с изм. и допол. в ред. от Стандартинформ, 2010
4. ГОСТ 19.503-79 ЕСПД. Руководство системного программиста. Требования к содержанию и оформлению (с Изменением N 1) от 01.01.1980 № 74 // январь 2010 г. с изм. и допол. в ред. от Стандартинформ, 2010
5. ГОСТ 19.504-79 ЕСПД. Руководство программиста. Требования к содержанию и оформлению (с Изменением N 1) от 01.01.1980 № 74 // январь 2010 г. с изм. и допол. в ред. от Стандартинформ, 2010
6. ГОСТ 19.505-79 ЕСПД. Руководство оператора. Требования к содержанию и оформлению (с Изменением N 1) от 01.01.1980 № 74 // январь 2010 г. с изм. и допол. в ред. от Стандартинформ, 2010
7. ГОСТ Р 53620-2009 Информационно-коммуникационные технологии в образовании. Электронные образовательные ресурсы. Общие положения от

01.01.2011 № 956-ст // 15.12.2009 г. с изм. и допол. в ред. от Стандартинформ, 2018

8. ГОСТ Р 52653-2006 от 01.07.2008 № 419-ст // Октябрь 2018 г. с изм. и допол. в ред. от Стандартинформ, 2018
9. ГОСТ Р 52657-2006. Информационно-коммуникационные технологии в образовании. Образовательные интернет-порталы федерального уровня. Рубрикация информационных ресурсов от 01.07.2008 № 423-ст // 2008 год г. с изм. и допол. в ред. от Стандартинформ, 2018
10. ГОСТ 34.602-89 Информационная технология (ИТ). Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы от 01.01.1990 № 661 // Июнь 2009 г. с изм. и допол. в ред. от Стандартинформ, 2009
11. Android Developers URL: https://developer.android.com/studio/ (дата обращения: 10.04.2019).
12. Android Studio // Википедия - свободная энциклоедия URL: https://ru.wikipedia.org/wiki/Android_Studio (дата обращения: 10.04.2019).
13. Anko // The world’s leading software development platform · GitHub URL: https://github.com/Kotlin/anko (дата обращения: 10.04.2019).
14. Create Android, iOS, and Windows apps with world-class tools // Mobile App Development & App Creation Software - Xamarin URL: https://www.xamarin.com/ (дата обращения: 10.04.2019).
15. DB Designer // Best database design URL: https://www.dbdesigner.net/ (дата обращения: 10.04.2019).
16. Download & Install // UML Designer Documentation URL: http://www.umldesigner.org/download/ (дата обращения: 10.04.2019).
17. Download and install jsoup // jsoup Java HTML Parser, with best of DOM, CSS, and jquery URL: https://jsoup.org/download (дата обращения: 10.04.2019).

18. Download Eclipse Technology that is right for you // Enabling Open Innovation & Collaboration | The Eclipse Foundation URL: http://www.eclipse.org/downloads/ (дата обращения: 10.04.2019).
19. Download IntelliJ IDEA // IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains URL: https://www.jetbrains.com/idea/download/ (дата обращения: 10.04.2019).
20. Downloads // Git URL: https://git-scm.com/download/ (дата обращения: 10.04.2019).
21. Draw Entity-Relationship Diagrams, Painlessly // dbdiagram.io - Database Relationship URL: https://dbdiagram.io/home (дата обращения: 10.04.2019).
22. Eclipse (среда разработки) // Википедия - свободная энциклопедия URL: https://ru.wikipedia.org/wiki/Eclipse_(среда_разработки) (дата обращения: 10.04.2019).
23. FlexboxLayout // The world’s leading software development platform · GitHub URL: https://github.com/google/flexbox-layout (дата обращения: 10.04.2019).
24. google-gson // The world’s leading software development platform · GitHub URL: https://github.com/google/gson (дата обращения: 10.04.2019).
25. IntelliJ IDEA // Википедия - свободная энциклопедия URL: https://ru.wikipedia.org/wiki/IntelliJ_IDEA (дата обращения: 10.04.2019).
26. NetBeans // Википедия - свободная энциклопедия URL: https://ru.wikipedia.org/wiki/NetBeans (дата обращения: 10.04.2019).
27. Ramus Educational // Software Informer - Windows software downloads and editorial reviews URL: http://ramus-educational.software.informer.com/ (дата обращения: 10.04.2019).
28. Roberto Ulloa Kivy - Interactive Applications and Games in Python. - 2 изд. Packt Publishing, 2015.
29. Rohit Ghatol, Yogesh Patel Beginning PhoneGap. - 1 изд. Apress, 2012.
30. SQL Database Modeler // SQL Database Modeler URL: https://sqldbm.com/Home/ (дата обращения: 10.04.2019).

31. Sylvain Ratabouil Android NDK Beginners Guide. - 2 изд. - Великобритания, Бирменгем: PACKT, 2015.
32. TIOBE Index for June 2019 // TIOBE Index | TIOBE - The Software Quality Company URL: https://www.tiobe.com/tiobe-index/ (дата обращения: 01.06.2019).
33. Williams Daniel Corona SDK Application Design. - 1 изд. - Великобритания, Бирмингем: PACKT, 2013.
34. Бесплатная пробная версия Adobe Photoshop CC // Adobe Россия: решения для творчества, маркетинга и работы с документами URL: https://www.adobe.com/ru/products/photoshop/free-trial-download.html (дата обращения: 10.04.2019).
35. Бонни Эйзенман Learning React Native: Building Native Mobile Apps with JavaScript. - 1 изд. - Себастопол: O'Reilly, 2015.
36. Джошуа Блох Java. Эффективное программирование. - М.: Лори, 2014.
37. Дмитрий Жемеров, Светлана Исакова Kotlin in Action. - 1 изд. - Шелтер Айленд, Нью-Йорк, США: MANNING, 2017.
38. Загрузить Java бесплатно // java.com: Java и вы URL: https://java.com/ru/download/ (дата обращения: 10.04.2019).
39. Загрузка среды NetBeans 8.2 // Welcome to NetBeans URL: https://netbeans.org/downloads/ (дата обращения: 10.04.2019).
40. Иванов А.А., Колташёва Д.Д., Сардак Л.В. Методические рекомендации по разработке дизайна в стиле Material Design для мобильного приложения под ОС Android // Акутальные вопросы преподавания математики, информатики и информационных технологий [Электронный ресурс] : Межвузовский сборник научных работ / Урал. гос. пед. ун-т; науч. ред. Л.В. Сардак – Электрон, дан. – Екатеринбург: [б.и.], 2019 – 1 электрон. опт. диск (CD-ROM). – С. 238-241.
41. Нативные приложения // Википедия - свободная энциклопендия URL: https://ru.wikipedia.org/wiki/Нативные_приложения (дата обращения: 10.04.2019).

42. Оформитель библиографических ссылок // SNOSKA.INFO URL: http://snoskainfo.ru/ (дата обращения: 14.02.2019).
43. Популярные среды разработки и их недостатки // GeekBrains - обучающий портал для программистов URL: https://geekbrains.ru/posts/ide_negative (дата обращения: 10.04.2019).
44. Проектирование баз данных // Википедия - свободная энциклопедия URL: https://ru.wikipedia.org/wiki/Проектирование_баз_данных (дата обращения: 10.04.2019).
45. Руководство по программированию для Xamarin Forms // METANIT.COM - Сайт о программировании URL: https://metanit.com/sharp/xamarin/ (дата обращения: 10.04.2019).
46. Скачайте Visual Studio // Visual Studio IDE, редактор кода, Team Services и Mobile Center URL: https://visualstudio.microsoft.com/ru/downloads/ (дата обращения: 10.04.2019).
47. Техническое задание на создание автоматизированной системы // Франклин&Грант: консалтинг, аналитика, Программное обеспечение, обучение для банков URL: https://www.franklin-grant.ru/ru/technologies/gost-34.602-89.shtml (дата обращения: 10.04.2019).
48. Эндрю Троелсен, Филипп Джепикс Язык программирования C# 6.0 и платформа .NET 4.6. - М.: Вильямс, 2016.