**Реферат**

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

*1*

*8*

*ДП 00.00.ПЗ*

*Разраб.*

*Славин Н.В.*

*Провер.*

*.*

*Сахонь Е.С.*

Консульт.

Сахонь Е.С.

*Н. Контр.*

Нистюк О.А.

*Утверд.*

*Смелов В. В.*

Реферат

*Лит.*

*Листов*

*1*

БГТУ 74217xxx, 2021

Пояснительная записка дипломного проекта содержит n страницы, n рисунков, n таблицы, n источников литературы, n приложений.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, КВЕСТ, ANDROID, JAVA, ANDROID STUDIO, RETROFIT, VISUAL STUDIO, ASP.NET CORE, C#, SQL, REST

Целью дипломного проекта является разработка мобильного приложения для выполнения заданий и получения вознаграждения «Квест».

В первой главе проводится аналитический обзор литературы и аналогичных решений по тематике дипломного проекта.

Вторая глава посвящена обзору средств разработки и содержит описание технологий, использованных во время выполнения проекта.

В третьей главе описывается процесс разработки, принципы функционирования и назначение созданных компонент проекта.

В четвертой главе описывается контрольный пример, с проведением тестирования, а также демонстрация поведения системы в различных внештатных ситуациях.

В пятой главе описано руководство пользователя, позволяющее подробно понять интерфейс программного средства.

В шестой главе приводится расчет экономических параметров и себестоимости программного средства, разработанного в рамках дипломного проекта.

Объем графической части дипломного проекта составляет n листов А1.

Реферат на иностранном

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

*1*

*8*

*ДП 00.00.ПЗ*

*Разраб.*

*Славин Н.В.*

*Провер.*

*.*

*Сахонь Е.С.*

Консульт.

Сахонь Е.С.

*Н. Контр.*

Нистюк О.А.

*Утверд.*

*Смелов В. В.*

Реферат

*Лит.*

*Листов*

*1*

БГТУ 74217xxx, 2021

**Содержание**

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

*1*

*8*

*ДП 00.00.ПЗ*

*Разраб.*

*Славин Н.В.*

*Провер.*

*.*

*Сахонь Е.С.*

Консульт.

Сахонь Е.С.

*Н. Контр.*

Нистюк О.А.

*Утверд.*

*Смелов В. В.*

Содержание

*Лит.*

*Листов*

*2*

БГТУ 74217xxx, 2021

[Введение 5](#_Toc71654119)

[1 Обзор теоретического материала и аналогичных решений 6](#_Toc71654120)

[1.1 Основные сведения по теме «Фриланс» 6](#_Toc71654121)

[1.2 Обзор фриланс-биржи «FL.ru» 7](#_Toc71654122)

[1.3 Обзор фриланс биржи «ALOT.PRO» 8](#_Toc71654123)

[1.4 Обзор фриланс биржи «Freelancer» 9](#_Toc71654124)

[1.5 Выводы по разделу 10](#_Toc71654125)

[2 Описание используемых технологий 12](#_Toc71654126)

[2.1 Язык программирования Java 12](#_Toc71654127)

[2.2 Платформа .NET 13](#_Toc71654128)

[2.3 Платформа ASP.NET Core 13](#_Toc71654129)

[2.4 СУБД Microsoft SQL Server 14](#_Toc71654130)

[2.5 Структурированный язык запросов SQL 15](#_Toc71654131)

[2.6 Технология Entity Framework Сore 16](#_Toc71654132)

[2.7 Язык программирования C# 17](#_Toc71654133)

[2.8 Выводы по разделу 18](#_Toc71654134)

[3 Проектирование и разработка программного средства 19](#_Toc71654135)

[3.1 Проектирование программного средства 19](#_Toc71654136)

[3.2 Файловая структура 19](#_Toc71654137)

[3.2.1 Структура Web-API 19](#_Toc71654138)

[3.2.2 Структура мобильного приложения 20](#_Toc71654139)

[3.3 Структура базы данных 21](#_Toc71654140)

[3.4 Разработка Web-API 23](#_Toc71654141)

[3.5 Разработка мобильного приложения 27](#_Toc71654142)

[3.7 Выводы по разделу 31](#_Toc71654143)

[4 Тестирование программного средства 32](#_Toc71654144)

[4.1 Авторизация пользователя 32](#_Toc71654145)

[4.2 Регистрация пользователя 33](#_Toc71654146)

[4.3 Создание квеста 34](#_Toc71654147)

[4.4 Изменение квеста 35](#_Toc71654148)

[4.5. Выводы по разделу 36](#_Toc71654149)

# Введение

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

*1*

*8*

*ДП 00.00.ПЗ*

*Разраб.*

*Славин Н.В.*

*Провер.*

*.*

*Сахонь Е.С.*

Консульт.

Сахонь Е.С.

*Н. Контр.*

Нистюк О.А.

*Утверд.*

*Смелов В. В.*

Введение

*Лит.*

*Листов*

*1*

БГТУ 74217xxx, 2021

Появление онлайн-бизнеса стало одним из важнейших этапов развития Интернета и бизнеса в целом. Благодаря E-commerce сформировалась новая рыночная система, кардинально отличающаяся от классической, и тысячи компаний по всей планете получили доступ к многомиллионной аудитории Сети.

Интернет превратился в универсальную платформу, позволяющую пользователям общаться, получать информацию, учиться и тратить деньги одновременно. Следующим логичным этапом его развития стала удалённая работа. Это способствовало появлению фриланса.

С 1972 года, когда идея удалённой работы была впервые озвучена, до недавнего времени фриланс оставался уделом веб-дизайнеров, программистов и т. п. Со временем, благодаря развитию технологий, в ряды фрилансеров также вошли переводчики, юристы, бухгалтеры и представители многих других профессий.

Платформами для удалённой работы стали биржи фриланса. Они могут стать для студентов первым источником дохода. Это позволяет в свободное от учёбы время не только получить дополнительный заработок, но и повысить практические навыки в рамках своей дисциплины. Для этого достаточно иметь при себе собственный ПК или ноутбук и постоянное интернет-соединение.

Преимущества удалённой работы. Предоставление возможности «набить руку» начинающим специалистам по самым разным направлениям: веб-разработка, администрирование сетей, дизайн, обработка фотографий, видеомонтаж. Кроме того, работу смогут найти и студенты более общих направлений, таких как интернет-реклама и копирайтинг. Биржи помогают пополнить портфолио примерами выполненных работ. На биржах нередко можно найти заказчика, который предложит сотрудничать на постоянной основе за более достойную оплату.

Целью данного дипломного проекта является создание мобильного приложения для выполнения заданий и получения вознаграждения «Квест».

Для достижения поставленной цели необходимо решить следующие задачи:

* спроектировать и создать базу данных;
* разработать мобильное приложение;
* разработать Web-API;
* обеспечить возможность регистрации и авторизации пользователей;
* обеспечить возможность создания, удаления и редактирования квестов;

# Обзор теоретического материала и аналогичных решений

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

*1*

*8*

*ДП 01.00.ПЗ*

*Разраб.*

*Славин Н.В.*

*Провер.*

*.*

*Сахонь Е.С.*

Консульт.

Сахонь Е.С.

*Н. Контр.*

Нистюк О.А.

*Утверд.*

*Смелов В. В.*

Обзор теоретического материала и аналогичных решений

*Лит.*

*Листов*

*6*

БГТУ 74217xxx, 2021

## Основные сведения по теме «Фриланс»

Фрилансер – свободный работник, который выполняет разовые задачи для разных заказчиков. Фрилансер чаще всего сам предлагает свои услуги – на специализированных онлайн-ресурсах, через газетные объявления или пользуясь «сарафанным радио», то есть личными связями.

Фриланс — это механизм, суть которого заключается в том, что некое частное лицо или фирма нанимает для выполнения определенной задачи человека, не зачисляя его в штат фирмы. Работник может находиться в другом городе или даже в другой стране, но может работать и в офисах заказчика. Широкое распространение фриланс получил с развитием интернета: сеть и сопутствующие информационные и банковские технологии позволили некоторым категориям работников уменьшить частоту появления в офисах, а то и полностью перейти на надомную/удалённую работу.

Чтобы найти работу фрилансером на дому, нужно просматривать вакансии на различных площадках. Это могут быть и доски объявлений, и биржи, и социальные сети.

В поисках заказов помогут доски объявлений. Например, Авито, Юла. Такой способ не очень эффективен, но онлайн ресурсы этого типа – неплохие места, где можно найти работу фрилансеру бесплатно, не выплачивая никаких комиссий. Со временем у специалиста устанавливается определенный процент заказов, которые он получает с подобных площадок. На таких сайтах можно найти постоянного заказчика.

Биржа – посредник между исполнителем и заказчиком. Она обеспечивает и первой, и второй стороне безопасную сделку и гарантированный результат. Для новичка биржи хороши тем, что на них можно найти первых постоянных клиентов и собрать портфолио.

Найти работу можно и на специализированных сайтах. Некоторые опытные фрилансеры и не думают о том, что найти работу можно на сайте HeadHunter. Здесь ищут не только продавцов в розничные магазины, но и копирайтеров, таргетологов, программистов. Фрилансер вправе составить резюме и выставить на этом сайте. Можно использовать и другие аналогичные ресурсы.

Когда у удаленного работника уже имеется опыт и портфолио, он может создать собственный сайт. Такая страничка станет визитной карточкой фрилансера. Важно постоянно работать над развитием сайта, наполнять его новыми готовыми работами, и по возможности создать блог на тематику фриланса.

## Обзор фриланс-биржи «FL.ru»

FL.ru – крупнейшая русскоязычная фриланс биржа. Она помогает найти талантливых исполнителей на любое задание. С помощью неё можно опубликовать бесплатное задание и выбрать исполнителя среди миллионов фрилансеров. Доступна возможность использования «Безопасной сделки» как удобного и легального способа работать с фрилансерами из других стран, как для организаций, так и для физических лиц. Также она гарантирует возврат денег, если исполнитель не справился с заказом или сорвал сроки.

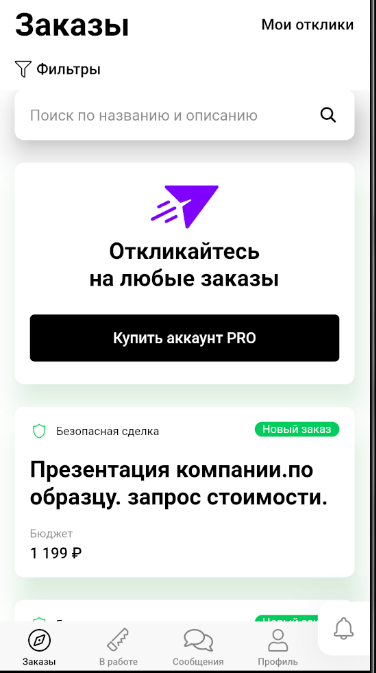


Рисунок 1.1 – Экран главной страницы в «FL.ru»

При регистрации приложение предлагает выбрать роль «Заказчик» или «Фрилансер». Для каждой роли отдельный аккаунт. Фрилансер может брать заказы, но для более лучших заказов потребуется купить PRO аккаунт. Та же ситуация и с заказчиком. Можно публиковать заказы, но для хороших заказов нужен PRO аккаунт.

К плюсам можно отнести удобный и понятный интерфейс, простоту использования и большое количество заказов.

К минусам относится то, что для хорошего заработка придётся покупать PRO аккаунт.

## Обзор фриланс биржи «ALOT.PRO»

Агрегатор вакансий Alot.Pro — продукт у которого нет аналогов. Начиная свой путь фриланса, люди тратят основную часть времени не на работу, а на её поиски. В подавляющем большинстве случаев это биржи фриланса и группы в социальных сетях.

Не имея отзывов, это делать ещё сложнее. Отсюда возникает проблема: чтобы иметь шансы на получение вакансии и конкурировать с другими фрилансерами, необходимо быстро реагировать на появившийся проект. Как следить за всеми сайтами сразу? Для этого и существует данная платформа, которая собирает данные со всех бирж и даже из групп «Вконтакте», чтобы вы первыми увидели свежую вакансию.

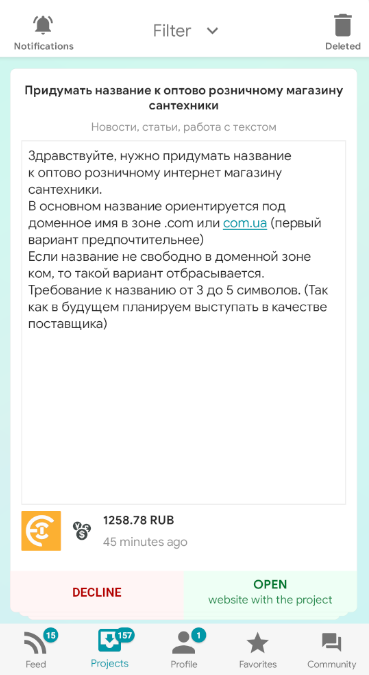


Рисунок 1.2 – Экран главной страницы в «ALOT.PRO»

Простота интерфейса как у мобильного приложения, так и у сайта позволяют сразу приступить к работе, без длительного изучения функционала.

Все собрано в 5 разделов:

1. Лента — раздел, где фрилансер найдет всю полезную информацию, свежие новости и подсказки по работе с сервисом
2. Проекты — здесь представлен весь список проектов, который формируется на фоне ваших фильтров.
3. Профиль — вся информация о вас, как о пользователи, а также, небольшой список достижений, за которые можно получить внутрисистемную валюту и потратить её на оплату Pro-версии.
4. Избранное — место, где хранятся все проекты, которые вы сюда поместили.
5. Форум — где вы всегда можете задать интересующий вас вопрос и быстро получить ответ.

Для работы на платформе вам не понадобиться тратить время на настройку и даже на регистрацию. Вам сразу предложат выбрать интересующие вас категории, после запуска приложения. После этого вы сразу сможете приступить к поиску вакансий. Если захотите синхронизировать аккаунты с разных устройств вам достаточно авторизоваться через любую из социальных сетей. Всё очень удобно.

Для доступа к сайтам проектов необходимо приобрести доступ. Стоимость месячной подписки - всего 265 рублей, а полного доступа навсегда - 1999 рублей. За эти деньги перед вами откроется возможность работать более чем с 20 сайтами в одном приложении, искать вакансии по ключевым словам и многое другое. Подключается в разделе «Профиль».

## Обзор фриланс биржи «Freelancer»

Freelancer, безусловно, является одним из крупнейших фриланс-сайтов в мире, и миллионы пользователей выбирали его на протяжении многих лет.

Функции и правила работы с сервисом Freelancer кому-то могут нравиться, кому-то — нет. Тем не менее, нельзя не согласиться с тем фактом, что именно у Freelancer они появились впервые, что именно Freelancer изобрел многое из того, что сегодня считается стандартом для фриланс-платформ. Тонны спама? Неадекватные отклики от неадекватных исполнителей? Freelancer по-прежнему процветает, оставляя не у дел менее “зубастых” конкурентов.

Различные баджи, уровни и сложная система опыта, используемые для ранжирования исполнителей, это уникальное изобретение Freelancer, которое многие другие сайты позаимствовали и улучшили, чтобы обойти Frelancer.com.

Для начинающего пользователя меню навигации сервиса Freelancer может показаться запутанным, однако на самом деле тут все организовано очень удобным образом. У Freelancer так много предложений для клиентов, что едва ли сервис мог обойтись меньшим количеством ссылок и кнопок. Регистрация очень простая, ее даже можно пропустить и авторизоваться с помощью учетной записи Facebook. Зарегистрироваться можно в качестве фрилансера, заказчика и в обеих этих ипостасях сразу.

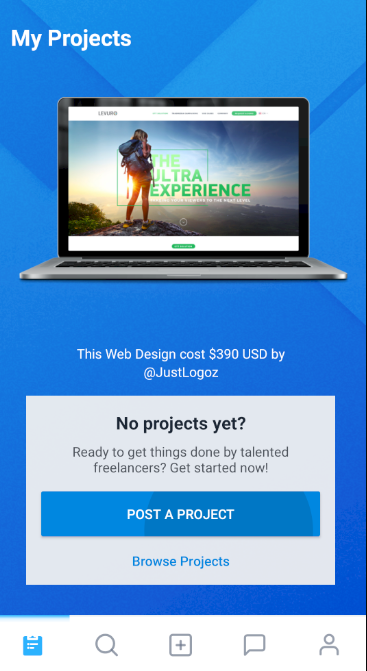


Рисунок 1.4 – Экран главной страницы в «Freelancer»

Вы начинаете работать с абсолютно бесплатным аккаунтом, но чтобы получить доступ к новым возможностям и уменьшить размер комиссии вам придется платить. Даже тесты для фрилансеров платные (от 5 долларов и выше). Вы можете заплатить, и тогда ваш проект или отклик на проект будет отображен на первом месте.

## Выводы по разделу

Данный раздел содержит информацию о аналогах разрабатываемого программного средства, был приведен краткий обзор к каждому из них, рассмотрены их возможности и недостатки, выделены плюсы и минусы.

Каждый аналог был подробно описан, и было показано, для чего предназначено то или иное программное обеспечение.

Как видно из представленного краткого обзора, большинство ресурсов имеют много общего в своих функциональных возможностях. В первую очередь это касается базовых функций: регистрация, создание заказов, взятие заказов в работу.

В итоге был сделан вывод, что некоторые особенности схожих по функционалу уже созданных программных средств недостаточны для достижения требуемого результата, в результате чего было принято решение на создание собственной разработки в рамках дипломного проекта.

# 2 Описание используемых технологий

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

*1*

*8*

*ДП 02.00.ПЗ*

*Разраб.*

*Славин Н.В.*

*Провер.*

*.*

*Сахонь Е.С.*

Консульт.

Сахонь Е.С.

*Н. Контр.*

Нистюк О.А.

*Утверд.*

*Смелов В. В.*

Описание используемых технологий

*Лит.*

*Листов*

*7*

БГТУ 74217xxx, 2021

## 2.1 Язык программирования Java

Java является объектно-ориентированным языком. Он поддерживает полиморфизм, наследование, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений.

На сегодняшний момент язык Java является одним из самых распространенных и популярных языков программирования. Java задумывался как универсальный язык программирования, который можно применять для различного рода задач. Текущей версией является Java 16, которая вышла в марте 2021 года. А Java превратилась из просто универсального языка в целую платформу и экосистему, которая объединяет различные технологии, используемые для целого ряда задач: от создания десктопных приложений до написания крупных веб-порталов и сервисов. Кроме того, язык Java активно применяется для создания программного обеспечения для множества устройств: обычных ПК, планшетов, смартфонов и мобильных телефонов и даже бытовой техники. Достаточно вспомнить популярность мобильной ОС Android, большинство программ для которой пишутся именно на Java.

Ключевой особенностью языка Java является то, что его код сначала транслируется в специальный байт-код, независимый от платформы. А затем этот байт-код выполняется виртуальной машиной JVM (Java Virtual Machine). В этом плане Java отличается от стандартных интерпретируемых языков как PHP или Perl, код которых сразу же выполняется интерпретатором. В то же время Java не является и чисто компилируемым языком, как С или С++.

Подобная архитектура обеспечивает кроссплатформенность и аппаратную переносимость программ на Java, благодаря чему подобные программы без перекомпиляции могут выполняться на различных платформах - Windows, Linux, Mac OS и т.д. Для каждой из платформ может быть своя реализация виртуальной машины JVM, но каждая из них может выполнять один и тот же код.

Java является языком с Си-подобным синтаксисом и близок в этом отношении к C/C++ и C#. Поэтому, если вы знакомы с одним из этих языков, то овладеть Java будет легче.

### 2.2 Платформа .NET

*.NET Framework* – это программная платформа, выпущенная компанией Microsoft, которая подходит для разных языков программирования.

.*NET* достаточно широко распространен в сфере разработки внутрикорпоративных программных продуктов, но в веб-разработке все же встречается относительно редко, как и другие программные продукты корпорации *Microsoft*. Поэтому найти разработчиков для веб-проекта бывает достаточно непросто. Использование. *NET* «тянет» за собой покупку и иного ПО от корпорации *Microsoft* (серверной ОС, СУБД и т.п.). Технология достаточно дорогая в разработке и сопровождении: кроме затрат на покупку лицензий на необходимое ПО существенный вклад в бюджет проектов вносят высокие зарплаты разработчиков.

*ASP.NET Core* – представляет технологию от компании *Microsoft*, предназначенную для создания различного рода веб-приложений: от небольших веб-сайтов до крупных веб-порталов и веб-сервисов.

### 2.3 Платформа ASP.NET Core

*ASP.NET Core* – свободно-распространяемый кросс-платформенный фреймворк для создания веб-приложений с открытым исходным кодом. Данная платформа разрабатывается компанией Майкрософт совместно с сообществом и имеет большую производительность по сравнению с *ASP.NET*.

Имеет модульную структуру и совместима с такими операционными системами как *Windows*, *Linux* и *MacOS*.

*ASP.NET Core* включает в себя фреймворк *MVC*, который объединяет функциональность *MVC*, *Web API* и *Web Pages*. В предыдущих версии платформы данные технологии реализовались отдельно и поэтому содержали много дублирующей функциональности. Сейчас же они объединены в одну программную модель *ASP.NET Core MVC*. А *Web Forms* полностью ушли в прошлое. Благодаря модульности фреймворка все необходимые компоненты веб-приложения могут загружаться как отдельные модули через пакетный менеджер *Nuget*. Кроме того, в отличие от предыдущих версий платформы нет необходимости использовать библиотеку *System.Web.dll*. Некоторые особенности *ASP.NET*:

* распространение пакетов платформы через *NuGet*;
* интегрированная поддержка для создания и использования пакетов *NuGet*;
* единый стек веб-разработки, сочетающий *Web UI* и *Web API*;
* конфигурация для упрощенного использования в облаке;
* встроенная поддержка для внедрения зависимостей;
* поддержка *CRUD*-операций при работе с таблицами через *GridView*;
* возможно создание веб-приложений, которые реализуют шаблон *Model-View-Controller* (*ASP.NET MVC Framework*).

### 2.4 СУБД Microsoft SQL Server

Microsoft SQL Server – система управления реляционными базами данных (СУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов – Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз, данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

Сервер баз, данных Microsoft SQL Server в качестве языка запросов использует версию языка SQL, получившую название Transact-SQL (сокращённо T-SQL). Язык T-SQL является реализацией SQL-92 (стандарт ISO для языка SQL) с множественными расширениями. T-SQL позволяет использовать дополнительный синтаксис для хранимых процедур и обеспечивает поддержку транзакций (взаимодействие базы данных с управляющим приложением).

SQL Server является одной из наиболее популярных систем управления базами данных (СУБД) в мире. Данная СУБД подходит для самых различных проектов: от небольших приложений до больших высоконагруженных проектов.

SQL Server был создан компанией Microsoft. Первая версия вышла в 1987 году. А текущей версией является версия 16, которая вышла в 2016 году и которая будет использоваться в текущем руководстве.

SQL Server долгое время был исключительно системой управления базами данных для Windows, однако начиная с версии 16 эта система доступна и на Linux.

SQL Server характеризуется такими особенностями как:

* SQL Server работает очень быстро;
* SQL Server предоставляет шифрование данных;
* с данной СУБД относительно легко работать и вести администрирование.

Центральным аспектом в MS SQL Server, как и в любой СУБД, является база данных. База данных представляет хранилище данных, организованных определенным способом. Нередко физически база данных представляет файл на жестком диске, хотя такое соответствие необязательно. Для хранения и администрирования баз, данных применяются системы управления базами данных (database management system) или СУБД (DBMS). И как раз MS SQL Server является одной из такой СУБД [4].

Для организации баз данных MS SQL Server использует реляционную модель. Эта модель баз данных была разработана еще в 1970 году Эдгаром Коддом. А на сегодняшний день она фактически является стандартом для организации баз данных.

Реляционная модель предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта

### 2.5 Структурированный язык запросов SQL

*SQL* (*Structured Query Language* – Структурированный язык запросов) – язык управления базами данных для реляционных баз данных. Сам по себе *SQL* не является тьюринг-полным языком программирования, но его стандарт позволяет создавать для него процедурные расширения, которые расширяют его функциональность до полноценного языка программирования.

Язык был создан в 1970х годах под названием «*SEQUEL*» для системы управления базами данных (СУБД) *System R*. Позднее он был переименован в «SQL» во избежание конфликта торговых марок. В 1979 году *SQL* был впервые опубликован в виде коммерческого продукта *Oracle V2*.

*SQL* создавался как простой стандартизированный способ извлечения и управления данными, содержащимися в реляционной базе данных. Позднее он стал сложнее, чем задумывался, и превратился в инструмент разработчика, а не конечного пользователя. В настоящее время SQL (по большей части в реализации Oracle) остается самым популярным из языков управления базами данных, хотя и существует ряд альтернатив.

*SQL* состоит из четырех отдельных частей.

Язык определения данных (*DDL*) используется для определения структур данных, хранящихся в базе данных. Операторы *DDL* позволяют создавать, изменять и удалять отдельные объекты в БД. Допустимые типы объектов зависят от используемой СУБД и обычно включают базы данных, пользователей, таблицы и ряд более мелких вспомогательных объектов, например, роли и индексы.

Язык манипуляции данными (*DML*) используется для извлечения и изменения данных в БД. Операторы *DML* позволяют извлекать, вставлять, изменять и удалять данные в таблицах. Иногда операторы select извлечения данных не рассматриваются как часть *DML*, поскольку они не изменяют состояние данных. Все операторы DML носят декларативный характер.

Язык определения доступа к данным (*DCL*) используется для контроля доступа к данным в БД. Операторы *DCL* применяются к привилегиям и позволяют выдавать и отбирать права на применение определенных операторов *DDL* и *DML* к определенным объектам БД.

Язык управления транзакциями (*TCL*) используется для контроля обработки транзакций в БД. Обычно операторы *TCL* включают commit для подтверждения изменений, сделанных в ходе транзакции, *rollback* для их отмены и savepoint для разбиения транзакции на несколько меньших частей.

### 2.6 Технология Entity Framework Сore

*Entity Framework Core* (EF Core) представляет собой объектно-ориентированную, легковесную и расширяемую технологию от компании *Microsoft* для доступа к данным. *EF Core* является *ORM*-инструментом (*object-relational mapping* - отображения данных на реальные объекты). То есть *EF Core* позволяет работать базами данных, но представляет собой более высокий уровень абстракции: *EF Core* позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает *Entity Framework*, мы уже работаем с объектами. Центральной концепцией *Entity Framework* является понятие сущности или *entity*. Сущность представляет набор данных, ассоциированных с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами. *Entity Framework Core* поддерживает множество различных систем баз данных. Таким образом, мы можем через *EF Core* работать с любой СУБД, если для нее имеется нужный провайдер.

По умолчанию на данный момент *Microsoft* предоставляет ряд встроенных провайдеров: для работы с *MS SQL Server*, для *SQLite*, для *PostgreSQL*. Также имеются провайдеры от сторонних поставщиков, например, для *MySQL*.

Также стоит отметить, что EF Core предоставляет универсальный API для работы с данными. И если, к примеру, мы решим сменить целевую СУБД, то основные изменения в проекте будут касаться прежде всего конфигурации и настройки подключения к соответствующим провайдерам. А код, который непосредственно работает с данными, получает данные, добавляет их в БД и т.д., останется прежним.

Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Например, если сущность описывает человека, то мы можем выделить такие свойства, как имя, фамилия, рост, возраст, вес. Свойства необязательно представляют простые данные типа int, но и могут представлять более комплексные структуры данных. И у каждой сущности может быть одно или несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность. Подобные свойства называют ключами.

При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Отличительной чертой *Entity Framework Core* является использование запросов *LINQ* для выборки данных из БД. С помощью *LINQ* мы можем не только извлекать определенные строки, хранящие объекты, из бд, но и получать объекты, связанные различными ассоциативными связями.

Другим ключевым понятием является *Entity Data Model*. Эта модель сопоставляет классы сущностей с реальными таблицами в БД.

*Entity Data Model* состоит из трех уровней: концептуального, уровень хранилища и уровень сопоставления (маппинга).

На концептуальном уровне происходит определение классов сущностей, используемых в приложении.

Уровень хранилища определяет таблицы, столбцы, отношения между таблицами и типы данных, с которыми сопоставляется используемая база данных.

Уровень сопоставления (маппинга) служит посредником между предыдущими двумя, определяя сопоставление между свойствами класса сущности и столбцами таблиц.

Таким образом, мы можем через классы, определенные в приложении, взаимодействовать с таблицами из базы данных.

*Entity Framework* предполагает три возможных способа взаимодействия с базой данных:

– *database first: Entity Framework* создает набор классов, которые отражают модель конкретной базы данных;

– *model first*: сначала разработчик создает модель базы данных, по которой затем *Entity Framework* создает реальную базу данных на сервере;

– *code firs*t: разработчик создает класс модели данных, которые будут храниться в бд, а затем *Entity Framework* по этой модели генерирует базу данных и ее таблицы.

### 2.7 Язык программирования C#

C# – язык программирования, сочетающий объектно-ориентированные и контекстно-ориентированные концепции. Разработан в 1998–2001 годах группой инженеров под руководством Андерсa Хейлсбергa в компании *Microsoft* как основной язык разработки приложений для платформы *Microsoft .NET*. Компилятор с C# входит в стандартную установку самой .*NET*, поэтому программы на нем можно создавать и компилировать даже без инструментальных средств вроде *Visual Studio.*

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и *Java*. Язык имеет строгую статическую типизацию, поддерживает полиморфизм, перегрузку операторов, указатели на функции-члены классов, атрибуты, события, свойства, исключения, комментарии в формате *XML*. Переняв многое от своих предшественников – языков C++, *Delphi, Modula* и *Smalltalk* – С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем: так, C# не поддерживает множественное наследование классов (в отличие от C++) или вывода типов (в отличие от *Haskell*). С помощью языка C# можно создавать обычные приложения *Windows*, *XML*-веб-службы, распределенные компоненты, приложения «клиент-сервер», приложения баз данных и т. д.

## 2.8 Выводы по разделу

В разделе проектирования были рассмотрены основные технологии, необходимые для разработки дипломного проекта. После изучения всех преимуществ этих технологий, для разработки программного средства были выбраны языки программирования С# и *Java* для написания мобильного приложения и *Web-API*. Данные языки способствуют качеству написания кода, отличаются широким спектром операционных систем, которые могут быть использованы для разработки, позволяют разрабатывать приложение с высоким уровнем быстродействия, а также отвечают всем современным стандартам качества разработки.

В качестве СУБД был выбран *MS SQL Server*, позволяющий в рамках проекта взаимодействовать с приложением. В рамках работы могут использоваться любые версии сервера, так как в *Entity Framework Core*, встроена работа со всеми вариантами данной базы данных.

# Проектирование и разработка программного средства

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

*1*

*8*

*ДП 03.00.ПЗ*

*Разраб.*

*Славин Н.В.*

*Провер.*

*.*

*Сахонь Е.С.*

Консульт.

Сахонь Е.С.

*Н. Контр.*

Нистюк О.А.

*Утверд.*

*Смелов В. В.*

3 Проектирования и разработка программного средства

*Лит.*

*Листов*

*13*

БГТУ 74217xxx, 2021

## Проектирование программного средства

Главной задачей данного дипломного проекта является создание мобильного приложения для выполнения заданий и получения вознаграждения для операционной системы *Android*. Исходя из изучения существующих аналогов были поставлены задачи по созданию интуитивно понятного интерфейса.

Функционально мобильное приложение должно выполнять следующие задачи:

* работа с пользователями (авторизация и регистрация);
* создание, удаление и редактирование квестов;
* отправка заявки на исполнение и выбор исполнителя для квеста;
* просмотр профилей пользователей;

Разрабатываемое программное средство представляет собой клиент-серверное приложение.

Клиент представляет собой приложение, установленное на смартфон с операционной системой *Android* с версией *API* 21 и выше. Клиент предназначен для взаимодействия пользователя с приложением и отправки запросов на сервер.

Сервер представляет собой *Web-API*, написанное на языке программирования C# с помощью платформы *ASP.NET* и *Entity Framework Core*. Сервер предназначен для обработки запросов, полученных от клиентов и отправки клиенту ответов.

## Файловая структура

### Структура Web-API

Перед началом разработки была продумана файловая структура проекта, позволившая быстрее ориентироваться в коде, вносить изменения, дописывать новый функционал. Скриншот файловой структуры *Web-API* представлен на рисунке 3.1.

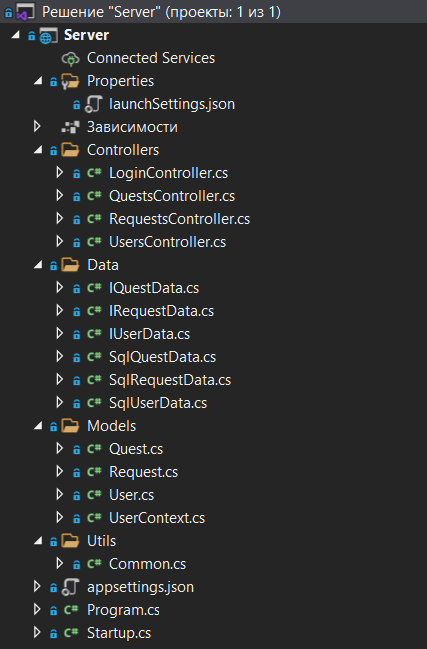


Рисунок 3.1 Файловая структура *Web-API*

Каталог *Models* содержит классы, которые описывают логику используемых данных. *Controllers* – каталог, который содержит файлы функционала приложения. *Data* содержит интерфейсы и классы для работы с БД. *Utils* содержит класс для хеширования паролей.

### Структура мобильного приложения

В данном подразделе я опишу структуру клиентского приложения и покажу некоторые особенности, обусловленные задачей приложения.

Основное решение проекта содержит большое количество папок и файлов, каждая из которых имеет свое назначение.

Скриншот файловой структуры мобильного приложения представлен на рисунке 3.2.

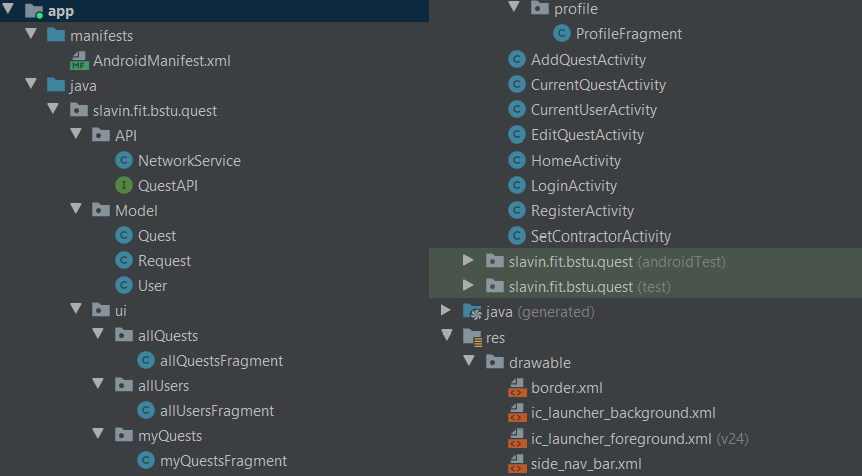


Рисунок 3.2 Файловая структура мобильного приложения

В основном здесь представлены файлы, описывающие поведение для активностей, но также присутствуют файлы для соединения и работы с сервером.

Каталог *Models* содержит классы, которые описывают логику используемых данных. Каталог *API* содержит в себе интерфейс и класс для соединения с *Web*-*API* и отправки запросов с помощью *Retrofit*. Каталог *ui* содержит в себе классы фрагментов для переходов по пунктам меню.

## Структура базы данных

В результате проектирования проекта была создана база данных, которая удовлетворяет поставленным задачам. Данная реализация достаточно проста и универсальна, с ее помощью в дальнейшем можно изменять и дополнять структуру базы данных. Для реализации функций, которые были определены на стадии проектирования, были созданы следующие таблицы:

* *Users* (информация о пользователях);
* *Quests* (квесты пользователей);
* *Requests* (запросы пользователей на исполнение квестов);
* *Images* (фотографии пользователей);

Логическая схема базы данных со всеми связями показана на рисунке 3.3 и в приложении А.

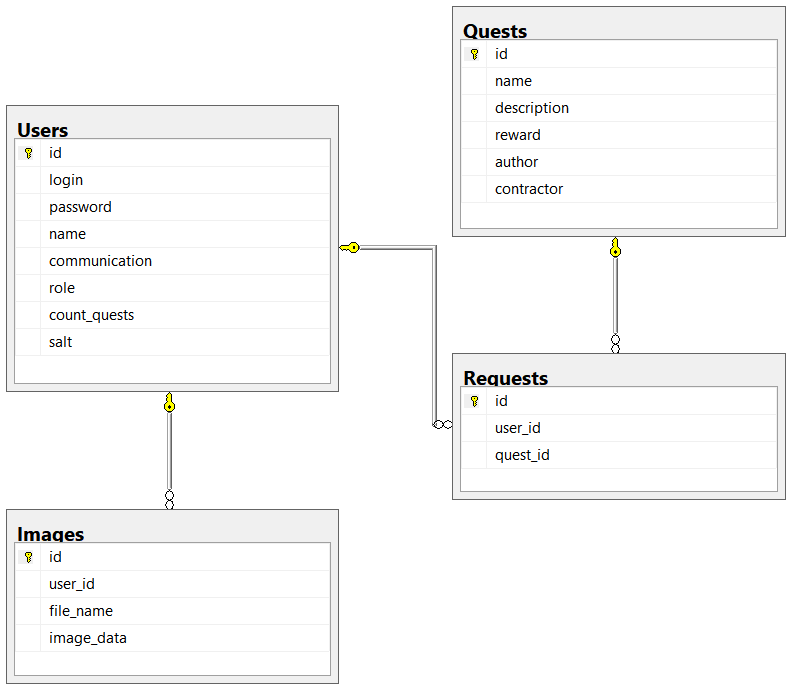


Рисунок 3.3 – Логическая схема базы данных

Для реализации функционала мобильного приложения было создано четыре таблицы. Далее приведена структура основных таблиц и их описание.

Таблица *Users* содержит *id* пользователя, логин, пароль, имя пользователя, способ связи с ним, роль, количество доступных квестов для создания и salt для хеширования.

Описание ее полей приведено в таблице 3.1.

Таблица 3.1 – Структура таблицы *Users*

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| id | int | Идентификатор пользователя |
| login | nvarchar(50) | Логин пользователя |
| password | nvarchar(50) | Захешированный пароль |
| name | nvarchar(50) | Имя пользователя |
| communication | nvarchar(250) | Способ связи |
| role | nvarchar(50) | Роль |
| count\_quests | int | Доступные квесты для создания |
| salt | nvarchar(250) | Хэш пароля |

Таблица *Quests* содержит идентификатор квеста, название квеста, описание квеста, вознаграждение за выполнение квеста, автора квеста, исполнителя квеста. Описание ее полей приведено в таблице 3.2.

Таблица 3.2 – Структура таблицы *Quests*

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| id | int | Идентификатор квеста |
| name | nvarchar(50) | Название квеста |
| description | nvarchar(500) | Описание квеста |
| reward | nvarchar(50) | Вознаграждение |
| author | nvarchar(50) | Логин автора |
| contractor | nvarchar(50) | Логин исполнителя |

Таблица *Requests* содержит идентификатор запроса, идентификатор квеста, идентификатор пользователя. Структура представлена в таблице 3.3.

Таблица 3.3 – Структура таблицы *Requests*

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| id | int | Идентификатор запроса |
| user\_id | int | Идентификатор квеста |
| quest\_id | int | Идентификатор пользователя |

Таблица *Images* содержит идентификатор фотографии, идентификатор пользователя, название файла, данные файла. Структура представлена в таблице 3.4.

Таблица 3.4 – Структура таблицы *Images*

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| id | int | Идентификатор фотографии |
| user\_id | Int | Идентификатор пользователя |
| file\_name | nvarchar(50) | Название файла |
| image\_data | varbinary(MAX) | Данные файла |

Выше были представлены таблицы, которые были созданы для функционирования всей системы. Логическая схема базы данных представлена в приложении А.

### 3.4 Разработка Web-API

Основной функционал сервера содержится в контроллерах. Это связано со спецификой разрабатываемой программы. Разберём основные контроллеры по порядку.

Первым разберём наиболее часто используемый контроллер *QuestsController*, который обеспечивает всю работу с квестами.

Пример программного кода контроллера представлен на листинге 3.1.

[ApiController]

public class QuestsController : ControllerBase

{

private IQuestData \_questData;

public QuestsController(IQuestData questData)

{

\_questData = questData;

}

[HttpGet]

[Route("api/[controller]")]

public string GetQuests()

{

return JsonConvert.SerializeObject(\_questData.GetQuests());

}

}

Листинг 3.1 – Программная реализация контроллера *QuestsController*

Основной функционал контроллера:

* получение списка квестов;
* добавление нового квеста;
* получение отдельного квеста;
* удаление отдельного квеста;
* изменение отдельного квеста.

Контроллер взаимодействует с базой данных с помощью запросов *LINQ to Entities*. Все запросы прописаны в классе *SqlQuestData*, который реализует интерфейс *IQuestData*.

Следующий по очереди идёт контроллер *UsersController*. Он обеспечивает нам работу с таблицей *Users*. Пример программного кода контроллера представлен на листинге 3.2.

[ApiController]

public class UsersController : ControllerBase

{

private IUserData \_userData;

public UsersController(IUserData userData)

{

\_userData = userData;

}

[HttpGet]

[Route("api/[controller]")]

public string GetUsers()

{

return JsonConvert.SerializeObject(\_userData.GetUsers());

}

}

Листинг 3.2 – Программная реализация контроллера *UsersController*

Основной функционал контроллера:

* получение списка пользователей;
* добавление нового пользователя;
* получение отдельного пользователя;
* удаление отдельного пользователя;
* изменение отдельного пользователя.

Контроллер взаимодействует с базой данных с помощью запросов *LINQ to Entities*. Все запросы прописаны в классе *SqlUserData*, который реализует интерфейс *IUserData*.

Определение всех действий с БД происходит в интерфейсах. В интерфейсе *IQuestData* определено 5 методов взаимодействия с данными. Эти методы представляют собой *CRUD*-операции, т.е. добавление, удаление, изменение и выборка данных. Программный код интерфейса представлен на листинге 3.3.

public interface IQuestData

{

List<Quest> GetQuests();

Quest GetQuest(int id);

string AddQuest(Quest quest);

void DeleteQuest(Quest quest);

Quest EditQuest(Quest quest);

}

Листинг 3.3 – Программный код интерфейса *IQuestData*

Все наши интерфейсы мы должны реализовать, чтобы иметь возможность работать с данными в нашей БД. Для каждого интерфейса мы создали файл, реализующий его. *SqlQuestData* – класс, реализующий интерфейс *IQuestData,* в котором у нас идёт реализация методов.

Программный код класса *SqlQuestData* представлен на листинге 3.4.

public class SqlQuestData : IQuestData

{

private UserContext \_userContext;

public SqlQuestData(UserContext userContext)

{

\_userContext = userContext;

}

public string AddQuest(Quest quest)

{

User user = \_userContext.Users.Where(u => u.login.Equals(quest.author)).First();

if (user.count\_Quests > 0)

{

\_userContext.Quests.Add(quest);

user.count\_Quests -= 1;

\_userContext.SaveChanges();

return "Квест успешно добавлен";

}

else

return "Вы достигли лимита квестов";

}

public void DeleteQuest(Quest quest)

{

\_userContext.Quests.Remove(quest);

\_userContext.SaveChanges();

}

public Quest EditQuest(Quest quest)

{

var existingQuest = \_userContext.Quests.Find(quest.id);

if (existingQuest != null)

{

existingQuest.name = quest.name;

existingQuest.description = quest.description;

existingQuest.reward = quest.reward;

existingQuest.contractor = quest.contractor;

\_userContext.Quests.Update(existingQuest);

\_userContext.SaveChanges();

}

return existingQuest;

}

public Quest GetQuest(int id)

{

var quest = \_userContext.Quests.Find(id);

return quest;

}

public List<Quest> GetQuests()

{

return \_userContext.Quests.ToList();

}

}

Листинг 3.4 – Программный код класса *SqlQuestData*

## 3.5 Разработка мобильного приложения

Основная логика программного средства реализована на стороне клиента, т.е. в мобильном приложении. Для взаимодействия клиентского приложения с сервером использовалась библиотека *Retrofit*. С помощью этой библиотеки можно отправлять запросы на сервер в асинхронном режиме. Для реализации этой библиотеки было создано несколько файлов. Первый файл – это интерфейс *QuestAPI*, который содержит в себе определения всех запросов, которые мы будет отправлять на сервер.

Пример программного кода интерфейса представлен на листинге 3.5.

public interface QuestAPI {  
 @GET("/api/quests/{id}")  
 @Headers("Content-Type: application/json")  
 public Call<Quest> GetQuest(@Path("id") int id);  
  
 @GET("/api/quests")  
 @Headers("Content-Type: application/json")  
 public Call<List<Quest>> GetQuests();  
  
 @DELETE("/api/quests/{id}")  
 @Headers("Content-Type: application/json")  
 public Call<String> DeleteQuest(@Path("id") int id);  
  
 @POST("/api/quests")  
 @Headers("Content-Type: application/json")  
 public Call<String> AddQuest(@Body Quest quest);  
  
 @PATCH("/api/quests/{id}")  
 @Headers("Content-Type: application/json")  
 public Call<String> EditQuest(@Path("id") int id, @Body Quest quest);

}

Листинг 3.5 – Программный код интерфейса *QuestAPI*

Следующий файл, который требовалось создать для реализации библиотеки *Retrofit* это *NetworkService*. С помощью него мы устанавливаем соединение с сервером и дальше отправляем нужный нам запрос и получаем ответ.

Пример программного кода интерфейса представлен на листинге 3.6.

public class NetworkService {  
 private static NetworkService *mInstance*;  
 private static final String *BASE\_URL* = "http://10.0.2.2:32024";  
 private Retrofit mRetrofit;  
  
 private NetworkService() {  
 HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor();  
 interceptor.setLevel(HttpLoggingInterceptor.Level.*BODY*);  
  
 OkHttpClient.Builder client = new OkHttpClient.Builder()  
 .addInterceptor(interceptor);  
  
 mRetrofit = new Retrofit.Builder()  
 .baseUrl(*BASE\_URL*)  
 .addConverterFactory(GsonConverterFactory.*create*())  
 .client(client.build())  
 .build();  
 }  
  
 public static NetworkService getInstance() {  
 if (*mInstance* == null) {  
 *mInstance* = new NetworkService();  
 }  
 return *mInstance*;  
 }  
  
 public QuestAPI getQuestApi() {  
 return mRetrofit.create(QuestAPI.class);  
 }  
}

Листинг 3.6 – Программная реализация класса *NetworkService*

Для отправки запросов на сервер часто вместе с запросом посылаются данные в виде объектов, с которыми нужно произвести какие-то действия. Модели данных размещены в папке *Model* и для каждой модели выделен отдельный файл.

Пример программного кода модели *Quest* представлен на листинге 3.7.

public class Quest {  
 @SerializedName("id")  
 private int id;  
 @SerializedName("name")  
 private String name;  
 @SerializedName("description")  
 private String description;  
 @SerializedName("reward")  
 private String reward;  
 @SerializedName("author")  
 private String author;  
 @SerializedName("contractor")  
 private String contractor;  
  
 public Quest() {  
  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getName() { return name; }  
  
 public void setName(String name) { this.name = name; }  
  
 public String getDescription() { return description; }  
  
 public void setDescription(String description) { this.description = description; }  
  
 public String getReward() { return reward; }  
  
 public void setReward(String reward) { this.reward = reward; }  
  
 public String getAuthor() { return author; }  
  
 public void setAuthor(String author) { this.author = author; }  
  
 public String getContractor() { return contractor; }  
  
 public void setContractor(String contractor) { this.contractor = contractor; }  
  
}

Листинг 3.7 – Программный код модели *Quest*

Оставшуюся часть файлов в мобильном приложении составляют файлы определения поведения активностей и их разметок. Файл *AddQuestActivity* – это активность, которая предназначена для создания нового квеста. В ней происходит создание объекта квеста и отправка *POST* запроса на сервер.

Пример программной реализации класса *AddQuestActivity* представлен на листинге 3.8.

public class AddQuestActivity extends AppCompatActivity {  
  
 Button add;  
 EditText name, description, reward;  
 String username;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_addquest*);  
  
 add = findViewById(R.id.*addQuest*);  
 name = findViewById(R.id.*name*);  
 description = findViewById(R.id.*description*);  
 reward = findViewById(R.id.*reward*);  
  
 ActionBar actionBar =getSupportActionBar();  
 actionBar.setHomeButtonEnabled(true);  
 actionBar.setDisplayHomeAsUpEnabled(true);  
  
 Bundle bundle = getIntent().getExtras();  
 username = bundle.getString("username");  
  
 add.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 add();  
 }  
 });  
 }  
  
 public void add() {  
 Quest quest = new Quest();  
 quest.setName(name.getText().toString());  
 quest.setDescription(description.getText().toString());  
 quest.setReward(reward.getText().toString());  
 quest.setContractor("none");  
 quest.setAuthor(username);  
  
 NetworkService.*getInstance*()  
 .getQuestApi()  
 .AddQuest(quest)  
 .enqueue(new Callback<String>() {  
 @Override  
 public void onResponse(@NonNull Call<String> call, @NonNull Response<String> response) {  
 Toast.*makeText*(AddQuestActivity.this, response.body(), Toast.*LENGTH\_LONG*).show();  
 if (response.body().equals("Квест успешно добавлен"))  
 back();  
 }  
  
 @Override  
 public void onFailure(@NonNull Call<String> call, @NonNull Throwable t) {  
 Toast.*makeText*(AddQuestActivity.this, t.getMessage(), Toast.*LENGTH\_LONG*).show();  
 t.printStackTrace();  
 }  
 });  
 }

Листинг 3.8 – Программная реализация активности *AddQuestActivity*

## 3.7 Выводы по разделу

В результате выполнения дипломного проекта было разработано мобильное приложение для выполнения заданий и получения вознаграждения «Квест». Все поставленные задачи перед проектом были выполнены.

Полученное мобильное приложение не требует специфического программного обеспечения, для его запуска необходимо наличие *Android API* в версии 21 и выше, а также доступ в сеть *Internet* любым доступным и удобным для пользователя способом. Проект является интуитивно понятным и простым. В программном средстве использованы устойчивые формы записи кода (использование обработчиков ошибок) благодаря чему повышается отказоустойчивость приложения.

Программное средство справляется со всеми поставленными задачами и целями, а именно:

* спроектирована и разработана структура базы данных;
* разработан интерфейс приложения;
* реализована авторизация и регистрация пользователей;
* реализован поиск пользователей и квестов;
* реализовано создание, редактирование и удаление квестов;
* разработаны контроллеры для обработки запросов клиентского приложения.

Спроектированная архитектура проекта, позволит добавлять и усовершенствовать функциональные возможности и поддерживать работу системы в будущем.

# Тестирование программного средства

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

*1*

*8*

*ДП 04.00.ПЗ*

*Разраб.*

*Славин Н.В.*

*Провер.*

*.*

*Сахонь Е.С.*

Консульт.

Сахонь Е.С.

*Н. Контр.*

Нистюк О.А.

*Утверд.*

*Смелов В. В.*

4 Тестирование программного средства

*Лит.*

*Листов*

*5*

БГТУ 74217xxx, 2021

Тестирование является неотъемлемой частью процесса разработки программного продукта, так как данный этап позволяет найти неисправности в работе приложения и исправить их.

В данной главе будут описаны некоторые тесты, проведенные для выявления и устранения неисправностей разработанного программного продукта.

Для начала представим названия тест-кейсов, которые и будем рассматривать ниже:

* авторизация пользователя;
* регистрация пользователя;
* создание квеста;
* изменение квеста;

## Авторизация пользователя

Для авторизации пользователю необходимо ввести свой логин и пароль, после чего нажать на кнопку «Войти». Поля не могут быть пустыми, иначе появится предупреждение.

После заполнения всех полей происходит запрос на сервер, на котором идёт поиск пользователя с введёнными данными. Если пользователь нашёлся, то нам приходит ответ с данными пользователя и происходит вход в приложение. Если пользователь не нашёлся, то выдаст ошибку, что такой пользователь не найден.

Скриншот тестирования авторизации представлен на рисунке 4.1.

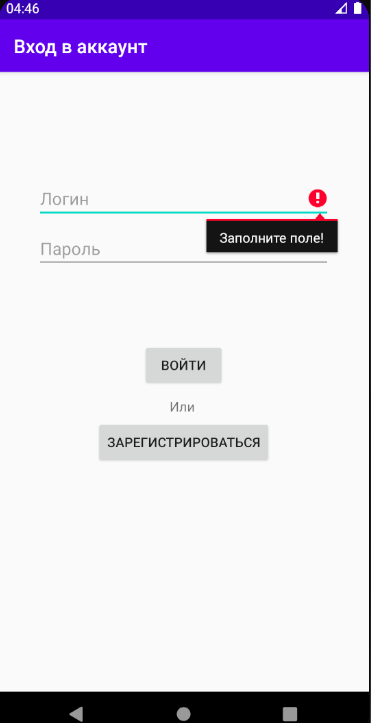


Рисунок 4.1 – Тестирование авторизации пользователя

## Регистрация пользователя

Для регистрации пользователю необходимо ввести логин, пароль, своё имя и способ связи с пользователем. Все поля не должны быть пустыми и длина логина и пароля должна составлять не менее 6 символов, иначе появится предупреждение.

Скриншот тестирования регистрации представлен на рисунке 4.2.

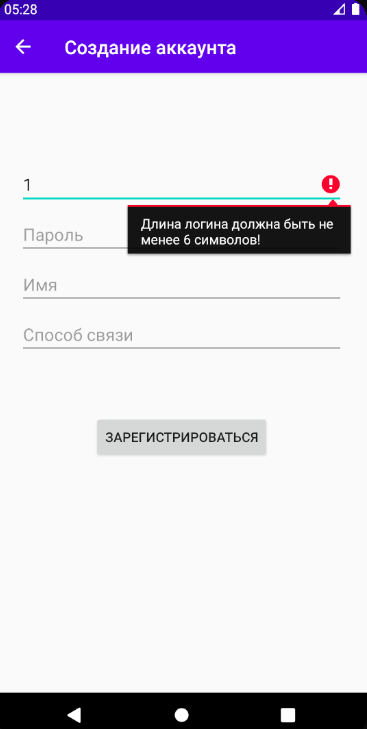


Рисунок 4.2 – Тестирование регистрации пользователя

После ввода всех данных нужно нажать на кнопку «Зарегистрироваться», тогда мы отправим запрос на сервер и если пользователь с таким логином не будет найдет, то аккаунт успешно будет создан.

## Создание квеста

Для создание квеста нам нужно ввести его название, описание и вознаграждение за его выполнение. Все поля не должны быть пустыми и имеют ограничение на максимальное количество символов.

Скриншот тестирования создания квеста представлен на рисунке 4.3.

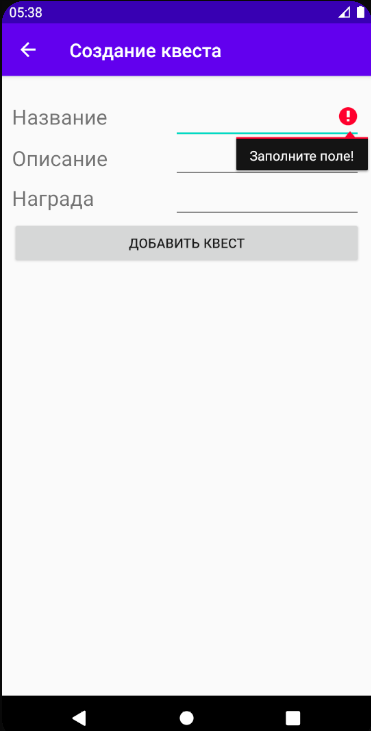


Рисунок 4.3 – Тестирование создания квеста

После успешного создания квеста, его можно будет найти в списке всех квестов или моих квестов.

## Изменение квеста

Для изменения квеста пользователю нужно изменить те данные, которые требуется и нажать на кнопку «Изменить квест». Тут действуют такие же ограничения, как и при создании квеста, а именно поля не должны быть пустыми и имеют ограничение на максимальное количество символов.

Скриншот тестирования изменения квеста представлен на рисунке 4.4.

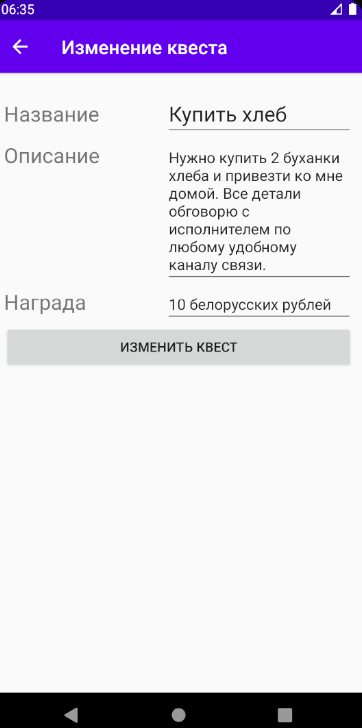


Рисунок 4.4 – Тестирование изменения квеста

## 4.5. Выводы по разделу

В данной главе был выполнен контрольный пример для обнаружения ошибок в поведении системы и выполнен ряд тестов для проверки ее стабильности. В результате проделанной работы установлено, что система стабильно работает, все тесты были выполнены успешно, а именно:

* авторизация пользователя;
* регистрация пользователя;
* создание квеста;
* изменение квеста;

В ходе тестирования было выявлено, что интерфейс обладаете достаточной функциональностью и удобством, а дизайн приложения имеет благоприятный для пользования вид.