МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Web-приложение для обмена сообщениями между пользователями информационной системы

Студент _____ Егинов Дмитрий Иванович
(подпись, дата) (ФИО полностью)

Руководитель _____ Князев В.Н._____
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В._____
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____№_____)

Заведующий кафедрой _____ _Макарычев П.П._____
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК _____ ____Попова Н.А._____
(подпись) (фамилия, инициалы)

Пенза, 2020

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____П.П. Макарычев
«30» декабря 2019 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Егинов Дмитрий Иванович гр. 16ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2.Тема работы Web-приложение для обмена сообщениями между пользователями информационной системы

Тема утверждена приказом ПГУ №_1488/о_от "_30__"_декабря_____2019___

3. Руководитель работы___к.т.н., доцент кафедры «МОиПЭВМ» Князев В.Н._____

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Разработать клиентскую и серверную части приложения по обмену сообщениями между пользователями информационной системы. Клиентская часть должна представлять собой мобильное приложение

2) Среда выполнения: операционная система Android 7.0 и выше для мобильного приложения, Unix-подобная операционная система для серверной части

Языковые средства разработки: Python, C#, UML

Инструментальные средства: PyCharm, Visual Studio 2019, Unity 2019.3.5, MongoDb, Draw.io, Figma, Postman, Simple Websocket Client, Smart Websocket Client

3) Технологии разработки: RUP, ООП, REST API, Websocket, DI

Основные функции web-приложения:

а) регистрация пользователей;

б) добавление другого пользователя в контакты;

в) отправка сообщений;

г) прием сообщений;

д) возможность создания групповых чатов;

е) отправка сообщений другому пользователю;

ж) шифрование сообщений.

5. Перечень подлежащих разработке вопросов

1) Анализ предметно области и постановка задачи

2) Проектирование

3) Реализация

4) Контроль качества программного обеспечения

5) Оформление пояснительной записки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	*Анализ предметной области и постановка задачи*	15%	13.04.2020 - 21.04.2020	
2	*Проектирование*	25%	22.04.2020 - 05.05.2020	
3	*Реализация*	40%	06.05.2020 - 30.05.2020	
4	*Контроль качества программного обеспечения*	5%	31.05.2020 - 02.06.2020	
5	*Оформление пояснительной записки*	15%	03.06.2020 - 13.06.2020	

Дата выдачи задания "30"__декабря_____2019___

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____
(подпись, дата)

Реферат

Пояснительная записка содержит 110 листов, 42 рисунка, 13 таблиц, 28 использованных источников, 5 приложений.

WEB-ПРИЛОЖЕНИЕ, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, МЕССЕНДЖЕР, СИСТЕМЫ ОБМЕНА СООБЩЕНИЯМИ, ANDROID, RUP, UML, PYTHON, C#, UNITY, REST, MONGODB

Объектом разработки является программное обеспечение web-приложения по обмену сообщениями между пользователями информационной системы.

Цель выпускной квалификационной работы – разработать серверную часть и мобильное приложение по обмену сообщениями.

Для разработки использовались инструментальные средства Microsoft Project, Draw.io, Figma. Проектирование производилось с использованием языка моделирования UML. Реализация компонентов приложения проводилась на языках Python и C# в средах разработки Unity, Visual Studio и PyCharm. Работа с базой данных осуществляется с помощью СУБД MongoDb.

В результате выполнения квалификационной работы разработано web-приложение для обмена сообщениями между пользователями информационной системы, отвечающее всем требованиям технического задания.

Содержание

Введение 7
1 Анализ предметной области и требований 9
1.1 Анализ предметной области и постановка задач 9
1.2 Анализ аналогичных программных продуктов на рынке систем обмена сообщениями 10
1.2.1 Анализ мобильного приложения WeChat 10
1.2.2 Анализ мобильного приложения ТамТам 12
1.2.3 Анализ мобильного приложения Skype 13
1.3 Анализ требований 14
1.3.1 Функциональные требования 14
1.3.2 Нефункциональные требования 21
1.4 Календарное планирование 22
2 Проектирование веб-приложения 35
2.1 Выбор технологий, инструментальных и языковых средств 35
2.2 Проектирование архитектуры программного обеспечения проекта 36
2.3 Проектирование интерфейса пользователя 37
2.4 Проектирование серверной части приложения 44
2.4.1 Проектирование базы данных 45
2.4.2 Проектирование API 47
3 Реализация веб-приложения 49
3.1 Реализация серверной части приложения 49
3.1.1 Реализация базы данных 49
3.1.2 Реализация серверной части приложения 49
3.2 Реализация мобильного приложения 51
4 Контроль качества программного обеспечения 55
4.1 Тестирование серверной части приложения 55
4.2. Тестирование клиентской части приложения 58
4.2.1 Функциональное тестирование 58
4.2.2 Метод эквивалентного разбиения 58
4.3 Метрики кода 59
4.3.1. Метрики Холстеда 59

4.3.2 Цикломатическая метрика МакКейба 62
4.4 Модульное тестирование 63
Заключение 66
Список использованных источников 67
Приложение А. Глоссарий 70
Приложение Б. Исходный код серверной части 72
Приложение В. Исходный код клиентской части 82
Приложение Г. Результаты тестирования серверной части 103
Приложение Д. Результаты тестирования мобильного приложения 107

Введение

В современном мире информационные технологии являются одной из наиболее развивающихся отраслей, им уделяется большое внимание как в России, так и во всем мире. Это подтверждается утверждением в 2017 году программы «Цифровая экономика Российской Федерации», основной задачей которой является «развитие цифровой экономики Российской Федерации, в которой данные в цифровой форме являются ключевым фактором производства во всех сферах экономической деятельности, что повышает конкурентоспособность страны...» [1].

Увеличение числа владельцев смартфонов, постоянное развитие технологий вызвали бурный рост интернет сервисов и мобильных приложений. «Стратегия развития информационного общества в Российской Федерации на 2017-2030 годы» предусматривает широкое применение сервисов сети Интернет и мобильных устройств и приложений [2].

Рынок мобильных приложений с каждым годом неуклонно растет. Это подтверждается данными аналитического агентства Digi-Capital – в 2019 году рынок вырос до $83.5 млрд, что на 21% выше показателя 2018 года [3]. Это обуславливается бурным развитием производства и продаж смартфонов и планшетов. Так, например, за 2019 год объем поставок смартфонов в Россию достиг 33 млн штук, что на 8% больше показателей 2018 года. В настоящее время невозможно найти сферу деятельности человека, в которой не применялись бы мобильные приложения. Они решают множество проблем и задач пользователей, одной из которых является обмен сообщениями.

Современный мир невозможно представить без систем обмена сообщениями. Ежесекундно обрабатываются миллионы запросов от пользователей на отправку и получение сообщений. Согласно исследованию Online Market Intelligence 83% пользователей смартфонов ежедневно используют мессенджеры в качестве систем обмена сообщениями [4]. Можно

сделать вывод, что в ближайшие годы этот рынок будет только увеличиваться. Современные мессенджеры являются технически сложными системами, с большим набором функционала. Зачастую пользователь не использует и половины возможностей таких приложений, из-за чего они оказываются перегружены функционалом, который только увеличивает нагрузку на смартфон. Также из-за этого увеличивается нагрузка на сервер приложения, что уменьшает скорость взаимодействия мобильного и веб-приложения в рамках информационной системы.

Исходя из вышесказанного целью выпускной квалификационной работы является актуальное направление по созданию веб-приложения, позволяющего пользователям информационной системы обмениваться сообщениями между собой.

1 Анализ предметной области и требований

1.1 Анализ предметной области и постановка задач

Разрабатываемое программное обеспечение предназначено для обмена сообщениями между пользователями. Согласно исследовании статистического агентства Statista за октябрь 2019 года за последние 2 года количество пользователей мессенджеров увеличилось на 75%. Число активных пользователей (в тыс. человек) самых популярных сервисов на октябрь 2019 года представлено на рисунке 1 [5].

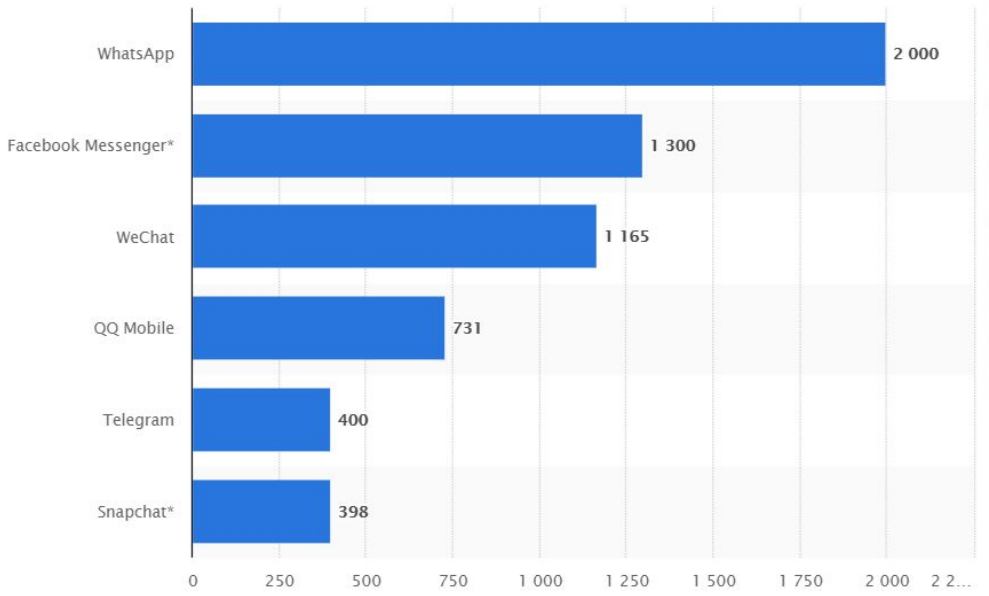

Рисунок 1 – Число активных пользователей (в тыс. человек) популярных мессенджеров на октябрь 2019 года

В современном мире человек непрерывно получает информацию, и поэтому важно, чтобы процесс обмена данными между людьми был надежным, безопасным, быстрым и простым. Актуальность тематики подчеркивается широким распространением таких систем обмена сообщениями, как мессенджеры, очень популярных на мобильных платформах.

Разрабатываемое приложение предназначено для того, чтобы пользователи могли обмениваться сообщениями друг с другом. Это обеспечит людей надежной системой обмена информацией.

Разрабатываемое приложение должно состоять из двух частей – клиентской и серверной. Приложение должно предоставлять пользователям следующие основные функции:

- регистрация пользователя;
- добавление другого пользователя в свои контакты;
- отправка сообщений;
- прием сообщений;
- возможность создания групповых чатов;
- отправка изображений другому пользователю;
- шифрование сообщений.

Для обеспечения безопасности сервер не должен сохранять историю переписок пользователей. Она должна храниться на устройстве, с возможностью резервного сохранения на сервере.

1.2 Анализ аналогичных программных продуктов на рынке систем обмена сообщениями

На данный момент существует множество сервисов, осуществляющих обмен сообщениями между пользователями. Из всего многообразия можно выделить следующие: WhatsApp, Telegram, Skype. Они являются лидерами на мировом рынке и значительно превосходят своих конкурентов.

1.2.1 Анализ мобильного приложения WeChat

WeChat был создан в 2011 году китайской компанией Tencent. Главный экран приведен на рисунке 2.

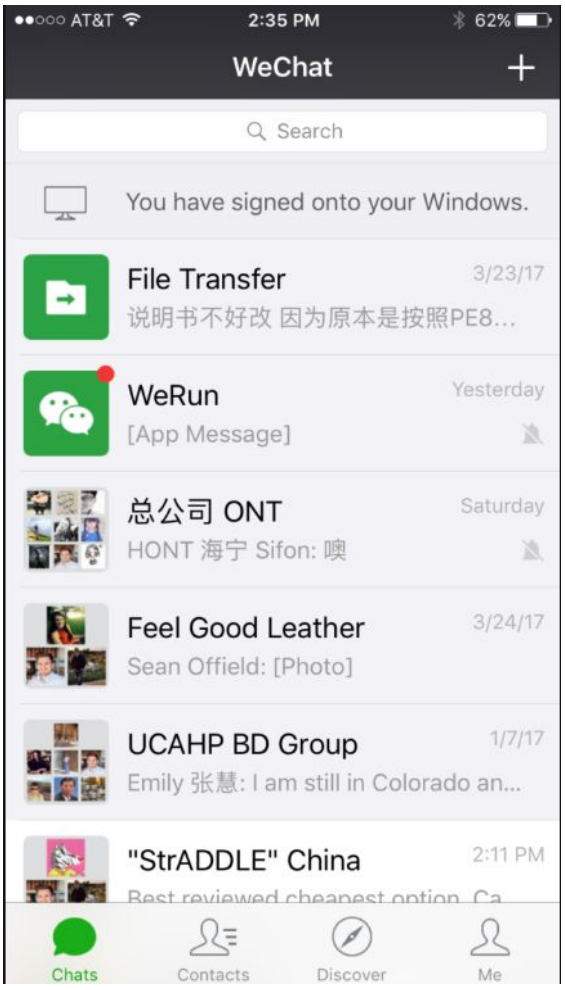

Рисунок 2 – Главный экран приложения WeChat

Приложение обладает простым и удобным интерфейсом, в нем содержится несколько разделов.

При работе с приложением можно выделить несколько основных достоинств и недостатков.

Из достоинств следует отметить то, что приложение полностью бесплатное. Еще одним достоинством является возможность совершения звонков от пользователя к пользователю.

Из недостатков можно выделить слабое шифрование, из-за чего персональные данные пользователя и его переписки могут быть украдены посторонними людьми. Также мобильное приложение не поддерживает работу на старых версиях операционных систем. Помимо этого, приложение

популярно на территории Китая, тогда как за его пределами оно не распространено.

1.2.2 Анализ мобильного приложения ТамТам

Telegram был создан в 2017 году отечественной компанией Mail.ru. Главный экран представлен на рисунке 3.

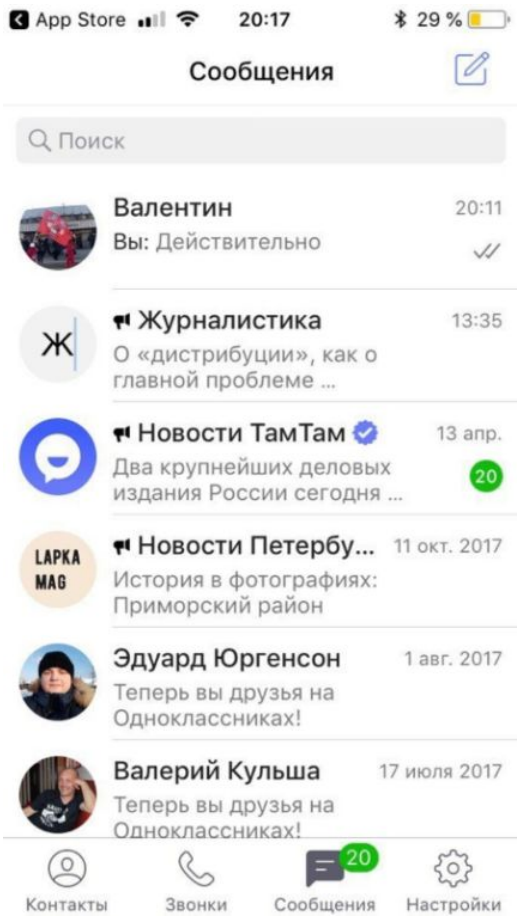

Рисунок 3 – Главный экран приложения ТамТам

Из достоинств следует отметить кроссплатформенность, приложение может работать на всех современных операционных системах, начиная от мобильных, заканчивая стационарными.

Главным недостатком является отсутствие открытой информации о том, как шифруются передаваемые данные. Из-за того, что система была разработана относительно недавно, она не поддерживается старыми версиями операционных систем. Также многие пользователи жалуются на

обилие рекламы и постоянные ошибки в работе приложения. Помимо этого, история переписок хранится на сервере, а не на устройстве, что уменьшает безопасность сохранения данных. Также приложение имеет популярность только на территории России, поэтому за ее пределами трудно найти пользователей приложения.

1.2.3 Анализ мобильного приложения Skype

Skype был создан в 2003 году компанией шведской компанией, а в 2011 году был куплен компанией Microsoft. Skype является первопроходцем в системах обмена сообщениями. Главный экран приведен на рисунке 4.

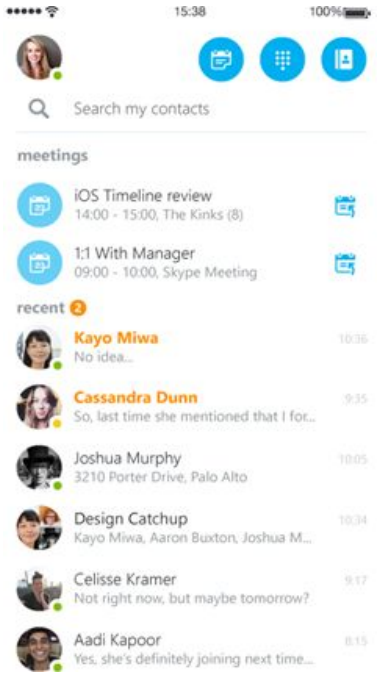

Рисунок 4 – Главный экран мобильного приложения Skype

Skype является старейшим приложением по обмену сообщениями, благодаря этому имеет большое количество пользователей. Skype предоставляет возможность совершения звонков и отправки SMS напрямую на мобильный телефон. Также в нем хорошо проработана система аудио и видеозвонков.

Главным недостатком Skype является недостаточная безопасность отправки данных. Многие пользователи жалуются на обилие рекламы и

постоянные ошибки в работе приложения. Отправить и получить сообщение возможно только в случае, если оба пользователя находятся в приложении или один из них использует облачное хранилище. Для того, чтобы воспользоваться полным функционалам, пользователю необходимо приобрести платную версию.

Таким образом, в результате анализа аналогов было принято решение о создании бесплатного сервиса для обмена сообщениями, лишенный недостатков аналогичных продуктов, который будет работать на операционных системах Android 7.0 и выше с дополнительным шифрованием данных и надежным сохранением истории переписок.

1.3 Анализ требований

Анализ требований – часть процесса разработки ПО, включающая в себя сбор требований, выявление зависимостей и их систематизацию и документирование.

Выделяют функциональные и нефункциональные требования. Функциональные требования – перечень задач, которые должно решать программное обеспечение. Они определяют действия, которые система должна выполнять, и как она должна реагировать на те или иные входные данные. Нефункциональные требования определяют свойства и характеристики системы, не относящиеся к ее поведению. Они включают производительность, расширяемость, надежность, безопасность и т.д.

1.3.1 Функциональные требования

В результате анализа предметной области и продуктов-аналогов были выделены следующие основные функциональные требования к системе:

- регистрация пользователя;
- добавление другого пользователя в свои контакты;
- отправка сообщений;

- прием сообщений;
- возможность создания групповых чатов;
- отправка изображений другому пользователю;
- шифрование сообщений.

Диаграмма вариантов использования представлена на рисунке 5.

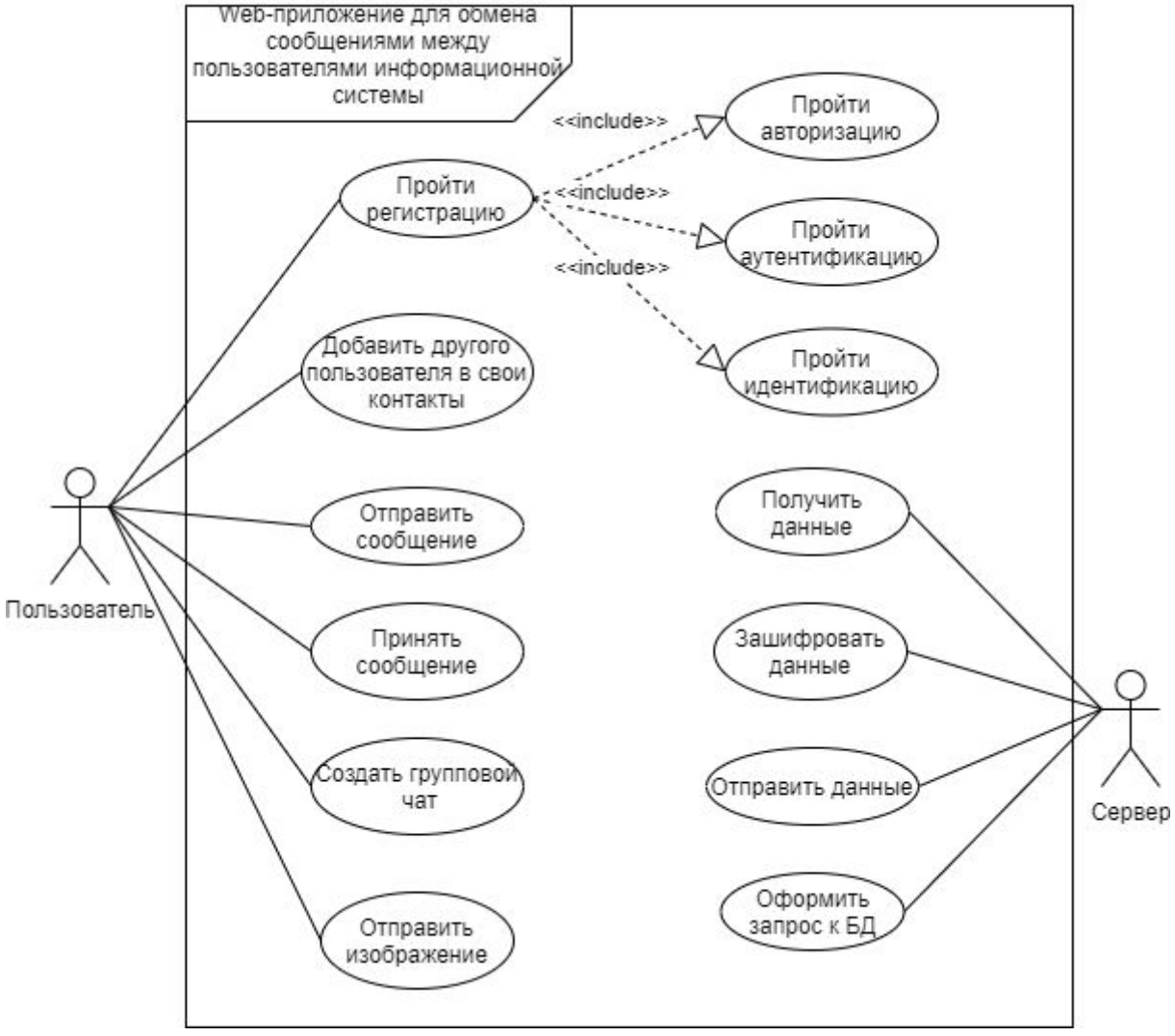

Рисунок 5 – Диаграмма вариантов использования

Данное описание прецедентов позволяет подробно рассмотреть работу некоторых вариантов использования.

Диаграмма вариантов использования отображает функциональные требования к системе. На диаграмме видно, какие функции выполняет каждый актер.

Рассмотрим подробнее некоторые прецеденты, для этого приведем их подробную спецификацию. На рисунке 6 приведена спецификация варианта использования «Принять сообщение».

Предецент: Принять сообщение
ID: 1
Краткое описание: Клиент1 получает сообщение от Клиента2
Главные актеры: Клиент1
Второстепенные актеры: Сервер, БД, Клиент2
Предусловия: Клиент2 отправил сообщение Клиенту1
Основной поток: 1. Сервер проверяет, находится ли Клиент1 онлайн 2. ЕСЛИ Клиент1 онлайн 2.1. Сервер отправляет сообщение Клиенту1 2.2. Приложение десериализует сообщение 2.3. Приложение отображает сообщение 3. ИНАЧЕ 3.1. Сервер сохраняет сообщение в БД до тех пор, пока Клиент1 не зайдет в приложение
Постусловия: Клиент1 получает сообщение
Альтернативные потоки: нет

Рисунок 6 – Спецификация прецедента «Принять сообщение»

Данная спецификация описана для варианта использования «Принять сообщение», который предназначен для обмена сообщениями между пользователями. Использование данного варианта позволит одному пользователю получить сообщение от другого, гарантируя доставку на устройство первого, независимо от того, находится ли он онлайн.

На рисунке 7 приведена спецификация варианта использования «Отправить сообщение».

Предецент: Отправить сообщение
ID: 2
Краткое описание: Сервер шифрует данные пользователя
Главные актеры: Клиент1, Клиент2
Второстепенные актеры: БД, Сервер
Предусловия: Клиент1 отправил сообщение для Клиента2
Основной поток: 1. Клиент1 отправляет сообщение 2. Сервер проверяет, находится ли Клиент2 онлайн 3. ЕСЛИ Клиент2 в приложении 3.1. Сервер отправляет сообщение Клиенту2 4. ИНАЧЕ 4.1. Сервер сохраняет сообщение в БД до тех пор, пока Клиент2 не зайдет в приложение
Постусловия: Второй клиент получает сообщение
Альтернативные потоки: нет

Рисунок 7 – Прецедент «Отправить сообщение»

Данная спецификация описана для варианта использования «Отправить сообщение», который предназначен для обмена сообщениями между пользователями. Использование данного варианта позволит одному пользователю отправить сообщение другому, гарантируя доставку на устройство второго, независимо от того, находится ли второй пользователь онлайн.

На рисунке 8 приведена спецификация варианта использования «Создать групповой чат».

Предецент: Создать групповой чат
ID: 3
Краткое описание: Клиент создает групповой чат с другими пользователями
Главные актеры: Клиент
Второстепенные актеры: Сервер, БД
Предусловия: Клиент открыл окно создания чата
Основной поток: 1. Клиент вводит ID или логины необходимых пользователей 2. Клиент нажимает кнопку «Создать» 3. Клиент отправляет запрос на сервер 4. Сервер создает запись о чате в базе данных 5. Сервер возвращает ответ об успешном создании чата 6. Мобильное приложение отображает новый чат в списке
Постусловия: У клиента появляется новый чат
Альтернативные потоки: нет

Рисунок 8 – Прецедент «Создать групповой чат»

Данная спецификация описана для варианта использования «Создать групповой чат», который предназначен для создания нового группового чата.

Для некоторых вариантов использования составим диаграммы деятельности. На рисунке 9 приведена диаграмма деятельности для прецедента «Отправить сообщение». Для того чтобы отправить сообщение отправитель должен ввести текст сообщения и нажать кнопку «отправить». Мобильное приложение шифрует текст с помощью алгоритма RSA, что

обеспечивает надежность и сохранность данных. После чего оно сохраняет текст в историю переписок, хранящуюся на устройстве, что позволит избежать утечек данных с сервера. Затем приложение, используя открытое ранее Websocket-соединение, посылает сообщение на сервер. Он, в свою очередь, проверяет, находится ли получатель в приложение. Если у получателя приложение закрыто, то сервер сохраняет сообщение в базе данных до тех пор, пока получатель не зайдет в приложение, в противном случае сервер отправляет полученное от отправителя сообщение на устройство получателя через Websocket-соединение. Мобильное приложение получателя дешифрует сообщение, сохраняет его в историю переписки и посылает пользователю оповещение о новом сообщении. Получатель сможет в любой момент перейти в диалог с отправителем и прочитать полученное сообщение.

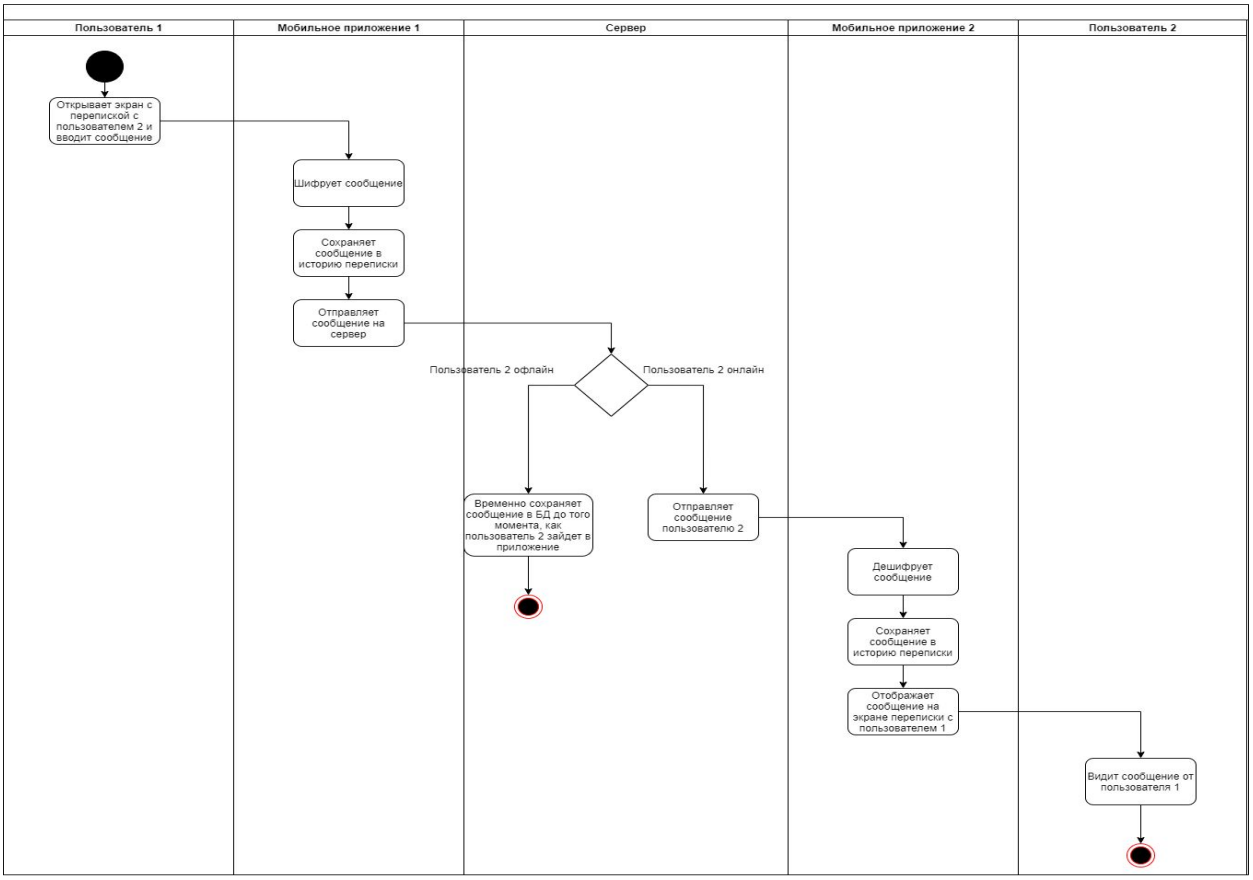

Рисунок 9 – Диаграмма деятельности для прецедента «Отправить сообщение»

На рисунке 10 приведена диаграмма деятельности для прецедента «Создать групповой чат». Для того чтобы создать групповой чат, пользователь должен ввести ID или логины всех участников и название чата. Сформируется и отправится POST запрос на сервер, который проверит корректность введенных данных, и в случае успеха создаст соответствующую запись в базе данных и вернет пользователю ответ об успешно выполненной операции.

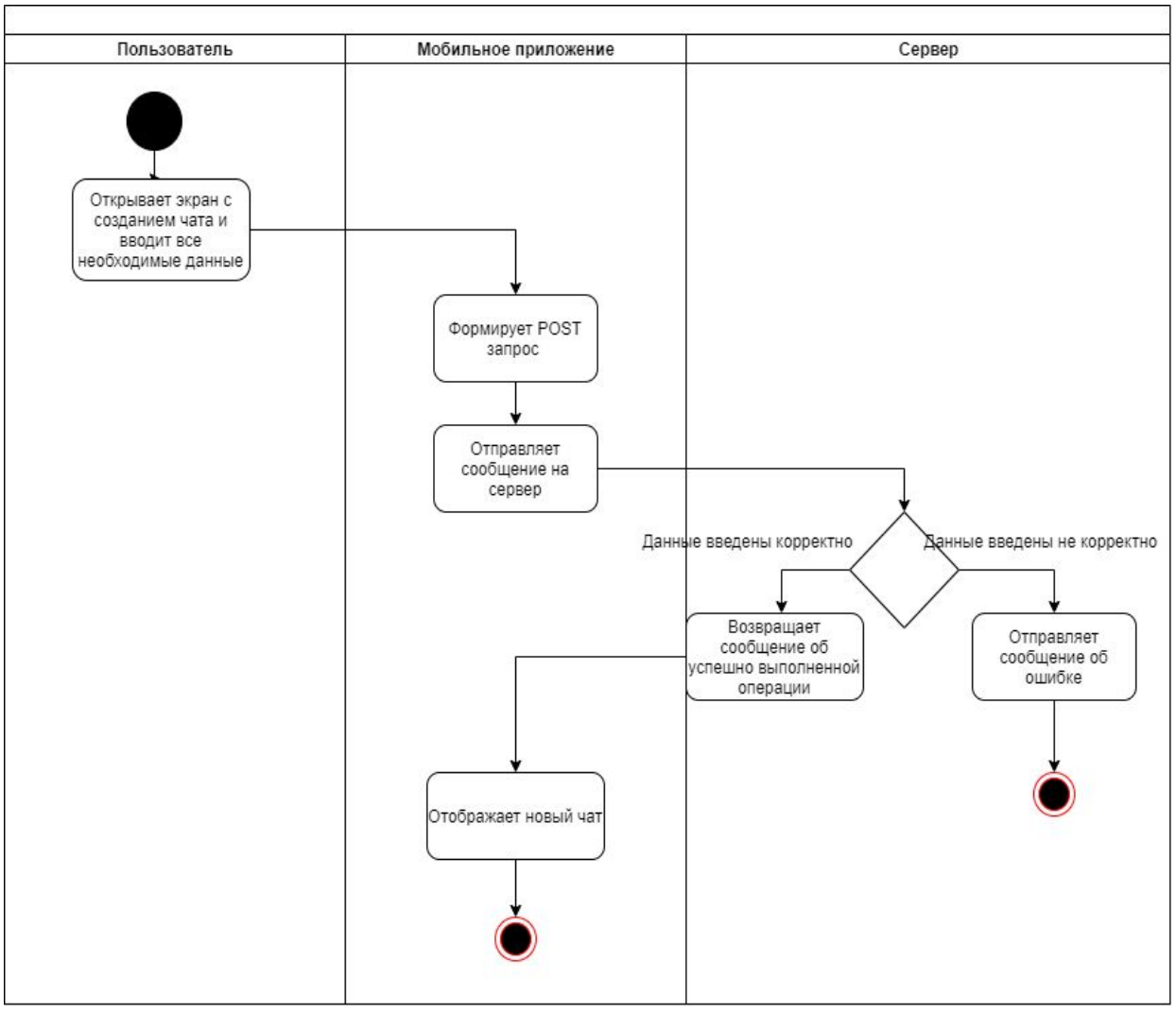

Рисунок 10 – Диаграмма деятельности для прецедента «Создать групповой чат»

На рисунке 11 приведена диаграмма последовательности для прецедента «Отправить сообщение».

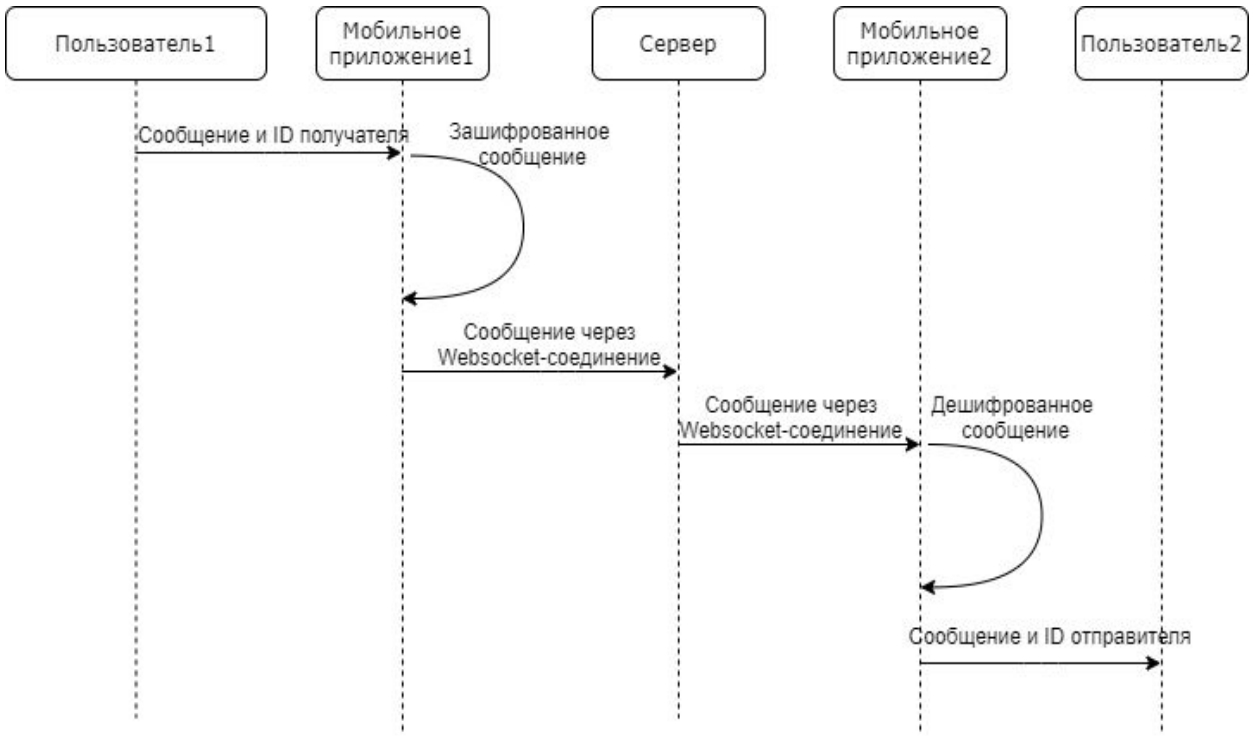

Рисунок 11 – Диаграмма последовательности для прецедента «Отправить сообщение»

В результате анализа функциональных требований были составлены требования к системе, разработана диаграмма вариантов использования и приведена спецификация некоторых прецедентов. Были разработаны диаграммы деятельности и последовательности для некоторых прецедентов.

1.3.2 Нефункциональные требования

Разработку можно разделить на 2 этапа: разработка серверной части и разработка клиентской части. Составим требования для каждой из них.

Приведем требования к клиентской части:

- клиентская часть должна является мобильным приложением;
- приложение должно обладать интерфейсом, удобным и понятным для пользователя;
- в приложении не должно возникать исключительных ситуаций, приводящих к аварийному завершению работы;

− разработка приложения должна осуществляться в кроссплатформенной среде разработки Unity 2019.3.5 и Visual Studio 2019 на языке программирования C# с использованием фреймворка UnityEngine;

− приложение должно функционировать на смартфонах с операционной системой Android 7.0 и выше;

− приложение должно быть расширяемым в случае появления новых функциональных возможностей.

Выделим требования к серверной части приложения:

− получение и обмен данными между пользователем и клиентом должно представлять из себя REST API;

− обмен сообщениями между пользователями должен осуществляться по протоколу Websocket;

− формат обмена данными между клиентом и сервером – формат JSON;

− разработка приложения должна осуществляться в среде PyCharm на языке программирования Python с использованием фреймворка Tornado;

− В качестве базы данных должна использоваться СУБД MongoDb.

В результате анализа были составлены нефункциональные требования к клиентской части и серверной части.

1.4 Календарное планирование и оценка бюджета проекта

Для определения трудоемкости и стоимости проекта необходимо провести его грамотное планирование и расчет бюджета. Для этого был определен список задач, решение которых необходимо для создания мобильного приложения, и перечень ресурсов, с помощью которых эти задачи будут решаться.

За начало проекта была принята дата 10.03.2020. В таблице 1 представлены задачи, представляющие из себя этапы создания проекта.

Таблица 1 – Перечень задач проекта

№	Название задачи	Длительность	Предшественники
1	Начало проекта	-	-
2	Анализ требований	-	-
3	Поиск литературы о системах обмена сообщениями	7 дней	1
4	Составление списка функциональных возможностей приложения	4 дня	3
5	Анализ протоколов клиент-серверного взаимодействия	3 дня	4
6	Анализ требований завершен	-	5
7	Выбор средств реализации проекта	-	-
8	Анализ возможных программных средств	5 дней	1
9	Выбор программного средства для реализации	2 дня	8
10	Выбор средств реализации завершен	-	9
11	Реализация проекта	-	-
12	Проектирование пользовательского интерфейса	4 дня	6;10
13	Проектирование структуры сообщений клиента и сервера	4 дня	6;10
14	Проектирование архитектуры сервера	4 дня	13
15	Реализация сервера	15 дней	14
16	Проектирование архитектуры клиента	3 дня	12,13
17	Реализация клиента	15 дней	16
18	Реализация безопасности передачи сообщений	7 дней	15;17
19	Реализация проекта завершена	-	18
20	Отладка	-	-
21	Тестирование работы отдельных модулей	5 дней	19
22	Тестирование работы классов	5 дней	19
23	Исправление ошибок	7 дней	21;22
24	Отладка завершена	-	23

Продолжение таблицы 1

№	Название задачи	Длительность	Предшественники
25	Внедрение программного проекта	-	-
26	Деплой сервера	2 дня	24
27	Загрузка клиента в GooglePlay	3 дня	24
28	Внедрение завершено	-	26;27
29	Конец проекта	-	28

Из этой таблицы можно сделать вывод, что для реализации программного проекта нужно пройти через решение 29 задач.

В таблице 2 представлен список затрачиваемых ресурсов, распределение которых приведено в таблице 3.

Таблица 2 – Список ресурсов проекта

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Программист1	Трудовой	150 Р\ч	300 Р\час
Программист2	Трудовой	150 Р\мес	300 Р\час
Интернет	Затраты		
Удаленный сервер	Затраты		
Компьютер	Материальный	30000 Р	
Android-устройство	Материальный	3000 Р	

В таблице 3 представлены назначения ресурсов.

Таблица 3 – Назначения ресурсов

№	Название задачи	Ресурс	Единицы
1	Начало проекта		
2	Анализ требований		
3	Поиск литературы о системах обмена сообщениями	Программист1 Компьютер Интернет	100% 1 шт

Продолжение таблицы 3

№	Название задачи	Ресурс	Единицы
4	Составление списка функциональных возможностей приложения	Программист1 Компьютер Интернет	100% 1 шт
5	Анализ протоколов клиент-серверного взаимодействия	Программист1 Компьютер Интернет	50% 100% 1 шт
6	Анализ требований завершен		
7	Выбор средств реализации проекта		
8	Анализ возможных программных средств	Программист2 Компьютер Интернет	100% 1 шт
9	Выбор программного средства для реализации	Программист2 Компьютер Интернет	100% 1 шт
10	Выбор средств реализации завершен		
11	Реализация проекта		
12	Проектирование пользовательского интерфейса	Программист1 Компьютер	100% 1 шт
13	Проектирование структуры сообщений клиента и сервера	Программист2 Компьютер	100% 1 шт
14	Проектирование архитектуры сервера	Программист1 Компьютер Интернет	100% 1 шт.
15	Реализация сервера	Программист1 Компьютер Интернет	100% 1 шт
16	Проектирование архитектуры клиента	Программист2 Компьютер Интернет	100% 1 шт

Окончание таблицы 3

№	Название задачи	Ресурс	Единицы
17	Реализация клиента	Программист2 Компьютер Интернет	100% 1 шт
18	Реализация безопасности передачи сообщений	Программист1 Программист2 Компьютер	50% 100% 2 шт
19	Реализация проекта завершена		
20	Отладка		
21	Тестирование работы отдельных модулей	Программист1 Компьютер Android-устройство	100% 1 шт 1 шт
22	Тестирование работы классов	Программист2 Компьютер Android-устройство	100% 1 шт 1 шт
23	Исправление ошибок	Программист1 Программист2 Компьютер Android-устройство	100% 100% 2 шт 1 шт
24	Отладка завершена		
25	Внедрение программного проекта		
26	Деплой сервера	Программист1	100%
27	Загрузка клиента в GooglePlay	Программист2 Компьютер Интернет Удаленный сервер	100% 1 шт.
28	Внедрение завершено		
29	Конец проекта		

Из представленных таблиц можно увидеть, что решением задач будут заниматься 2 программиста, которым для этого необходимы компьютеры,

доступ к интернету, Android-устройству для тестирования приложения и удаленный сервер для развертывания серверной части приложения.

Сначала вручную просчитаем время, которое необходимо для реализации проекта и определим его критический путь. Для этого необходимо построить сетевой график проекта [6]. Он представлен на рисунке 12.

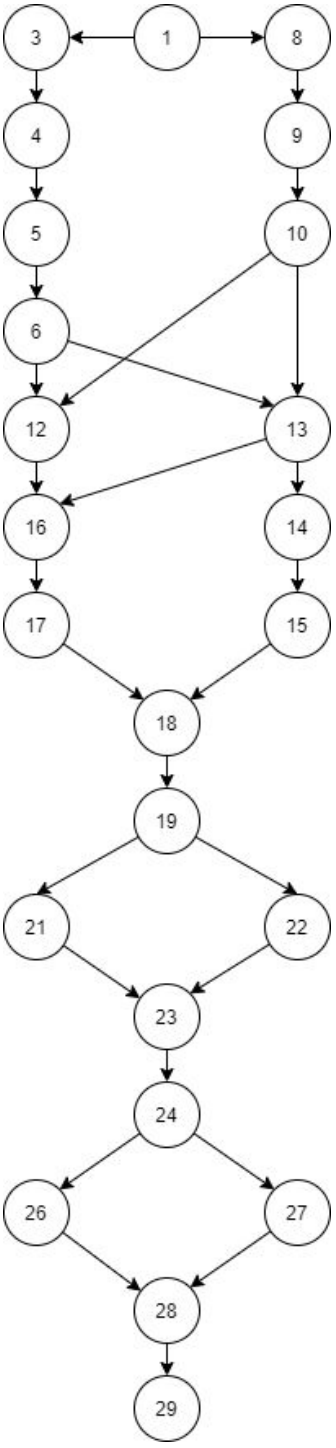

Рисунок 12 – Сетевой график проекта

Обозначим через t_i время выполнения работы i, G – множество работ, непосредственно предшествующих работе i, H – множество работ, непосредственно следующих за работой i [7].

Для вычисления критического пути необходимо для каждой задачи определить раннее и позднее время начала работ. Ранее время определяется по формуле:

$$T_{\text{р}(i)} = \left(T_{\text{р}(j)} + t_j \right).$$

Позднее время начала следует вычислять по формуле:

$$T_{\text{п}(i)} = \left(T_{\text{п}(j)} - t_j \right).$$

Вычисление раннего времени начала работ расписано в таблице 4.

Обозначим через t_i время выполнения работы i, G – множество работ, предшествующих работе i, H – множество работ, следующих за работой i. Для вычисления критического пути необходимо для каждой задачи определить раннее и позднее время начала работ. Ранее время определяется по формуле:

$$T_{\text{р}(i)} = \left(T_{\text{р}(j)} + t_j \right).$$

Позднее время начала следует вычислять по формуле:

$$T_{\text{п}(i)} = \left(T_{\text{п}(j)} - t_j \right).$$

Вычисление раннего времени начала работ расписано в таблице 4.

Таблица 4 – Вычисление раннего времени начала работ

Работа	Предшественники			Вычисление	Раннее время начала
	№	Раннее время начала	Длительность		
1	-	-	-	-	0
3	1	0	0	0+0	0
4	3	0	7	0+7	7

Продолжение таблицы 4

Работа	Предшественники			Вычисление	Раннее время начала
	№	Раннее время начала	Длительность		
5	4	7	4	7+4	11
6	5	11	3	11+3	14
8	1	0	0	0+0	0
9	8	0	5	0+5	5
10	9	5	2	5+2	7
12	6 10	14 7	0 0	Max(14+0;7+0)	14
13	6 10	14 7	0 0	Max(14+0;7+0)	14
14	13	14	4	14+4	18
15	14	18	4	18+4	22
16	12 13	14 14	4 4	Max(14+4;14+4)	18
17	16	18	3	18+3	21
18	15 17	22 21	15 15	Max(22+15;21+15)	37
19	18	37	7	37+7	44
21	19	44	0	44+0	44
22	19	44	0	44+0	44
23	21 22	44 44	5 5	Max(44+5;44+5)	49
24	23	49	7	49+7	56
26	24	56	0	56+0	56
27	24	56	0	56+0	56
28	26 27	56 56	2 3	Max(56+2;56+3)	59
29	28	59	0	59+0	59

Из этих вычислений можно увидеть, что выполнение проекта должно занять 59 рабочих дней. Теперь необходимо вычислить время позднего начала работ. Оно представлено в таблице 5.

Таблица 5 – Вычисление позднего времени начала работ

Работа	Последователи		Длительность	Вычисление	Позднее время начала
	№	Позднее время начала			
29	-	-	-	-	59
28	29	59	0	59-0	59
27	28	59	3	59-3	56
26	28	59	2	59-2	57
24	26 27	56 57	0	Min(65-0;57-0)	56
23	24	56	7	56-7	49
22	23	49	5	49-5	44
21	23	49	5	49-5	44
19	21 22	44 44	0	Min(44-0;44-0)	44
18	19	44	7	44-7	37
17	18	37	15	37-15	22
16	17	22	3	22-3	19
15	18	37	15	37-15	22
14	15	22	4	22-4	18
13	14 16	18 19	4	Min(18-4;19-4)	14
12	16	19	4	19-4	15
10	12 13	15 14	0	Min(15-0;14-0)	14
9	10	14	2	14-2	12
8	9	12	5	12-5	7

Продолжение таблицы 5

Работа	Последователи		Длительность	Вычисление	Позднее время начала
	№	Позднее время начала			
6	12 13	15 14	0	Min(15-0;14-0)	14
5	6	14	3	14-3	11
4	5	11	4	11-4	7
3	4	7	7	7-7	0
1	3,8	0,7	0	Min(0-0;7-0)	0

Зная оба вида работ, можно определить критический путь для нашего проекта. Все необходимые для этого вычисления представлены в таблице 6.

Таблица 6 – Вычисление критического пути

Работа №	Раннее время начала	Позднее время начала	Резерв времени
1	0	0	0
3	0	0	0
4	7	7	0
5	11	11	0
6	14	14	0
8	0	7	7
9	5	12	7
10	7	14	7
12	14	15	1
13	14	14	0
14	18	18	0
15	22	22	0
16	18	19	1
17	21	22	1
18	37	37	0
19	44	44	0
21	44	44	0

Продолжение таблицы 6

Работа №	Раннее время начала	Позднее время начала	Резерв времени
22	44	44	0
23	49	49	0
24	56	56	0
26	56	57	1
27	56	56	0
28	59	59	0
29	59	59	0

Из этой таблицы можно определить, что критический путь проходит через работы, с номерами 1, 3, 4, 5, 6, 13, 14, 15, 18, 19, 21, 22, 23, 24, 27, 28, 29. Его можно наглядно проследить на рисунке 13.

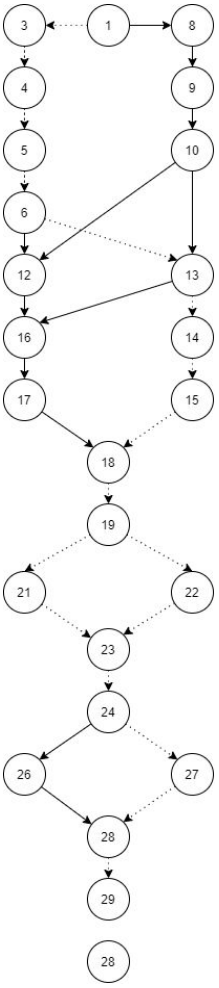

Рисунок 13 – Критический путь

Теперь после выбора программных средств планирования можно провести все необходимые вычисления с помощью системы управления проектами Microsoft Project. За начало работ была принята дата 10.03.2020. Перечень задач проекта представлен на рисунке 14 [8].

		Название задачи	Длительность	Начало	Окончание
1		Начало работ	0 дней	Вт 10.03.20	Вт 10.03.20
2		**Анализ требований**	**14 дней**	**Вт 10.03.20**	**Пт 27.03.20**
3		Поиск литературы о систем	7 дней	Вт 10.03.20	Ср 18.03.20
4		Составление списка функц	4 дней	Чт 19.03.20	Вт 24.03.20
5		Анализ протоколов клиент-	3 дней	Ср 25.03.20	Пт 27.03.20
6		Анализ требований заверш	0 дней	Пт 27.03.20	Пт 27.03.20
7		**Выбор средств реализации п**	**7 дней**	**Вт 10.03.20**	**Ср 18.03.20**
8		Анализ возможных програ	5 дней	Вт 10.03.20	Пн 16.03.20
9		Выбор программного сред	2 дней	Вт 17.03.20	Ср 18.03.20
10		Выбор средств реализаци	0 дней	Ср 18.03.20	Ср 18.03.20
11		**Реализация проекта**	**30 дней**	**Пн 30.03.20**	**Пт 08.05.20**
12		Проектирование пользоват	4 дней	Пн 30.03.20	Чт 02.04.20
13		Проектирование структуры	4 дней	Пн 30.03.20	Чт 02.04.20
14		Проектирование архитекту	4 дней	Пт 03.04.20	Ср 08.04.20
15		Реализация сервера	15 дней	Чт 09.04.20	Ср 29.04.20
16		Проектирование архитекту	3 дней	Пт 03.04.20	Вт 07.04.20
17		Реализация клиента	15 дней	Ср 08.04.20	Вт 28.04.20
18		Реализация безопасности	7 дней	Чт 30.04.20	Пт 08.05.20
19		Реализация проекта завер	0 дней	Пт 08.05.20	Пт 08.05.20
20		**Отладка**	**12 дней**	**Пн 11.05.20**	**Вт 26.05.20**
21		Тестирование работы отде	5 дней	Пн 11.05.20	Пт 15.05.20
22		Тестирование работы клас	5 дней	Пн 11.05.20	Пт 15.05.20
23		Исправление ошибок	7 дней	Пн 18.05.20	Вт 26.05.20
24		Отладка завершена	0 дней	Вт 26.05.20	Вт 26.05.20
25		**Внедрение программного про**	**3 дней**	**Ср 27.05.20**	**Пт 29.05.20**
26		Деплой сервера	2 дней	Ср 27.05.20	Чт 28.05.20
27		Загрузка клиента в Google	3 дней	Ср 27.05.20	Пт 29.05.20
28		Внедрение завершено	0 дней	Пт 29.05.20	Пт 29.05.20
29		Конец проекта	0 дней	Пт 29.05.20	Пт 29.05.20

Рисунок 14 – Задачи проекта, введенные в среду Microsoft Project

Диаграмма Ганта изображена на рисунке 15.

Рисунок 15 – Диаграмма Ганта

Далее был получен обзор на проект веб-приложения для обмена сообщениями между пользователями информационной системы. Результат представлен на рисунке 16.

ПроектНир
НР

на дату Вт 21.04.20

Даты			
Начало:	Вт 10.03.20	Окончание:	Пт 29.05.20
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней

Длительность			
Текущий план:	59 дней	Оставшаяся:	59 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	59 дней	Процент завершени	0%

Трудозатраты			
Текущий план:	788 ч	Оставшиеся:	788 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	788 ч	Процент завершени	0%

Затраты			
Текущий план:	667 200,00 ₽	Оставшиеся:	667 200,00 ₽
Базовый план:	0,00 ₽	Фактические:	0,00 ₽
Отклонение:	667 200,00 ₽		

Состояние задачи	
Еще не начатые задачи:	29
Выполняющиеся задачи	0
Завершенные задачи:	0
Итого задач:	29

Состояние ресурса	
Трудовые ресурсы:	4
Трудовые ресурсы с прев. дост	0
Материальные ресурсы:	2
Итого ресурсов:	6

Рисунок 16 – Обзор проекта

Отчет показал, что длительность реализации проекта будет равна 59 дням, а предполагаемые затраты составят 667 200 р.

2 Проектирование веб-приложения

2.1 Выбор технологий, инструментальных и языковых средств разработки

Технология RUP (Rational Unified Process) – ведущая методология разработки ПО, в которой инструментально поддерживаются все этапы жизненного цикла. В основе методологии лежит принцип пошаговой разработки продукта. Для каждого этапа определяются состав и последовательность задач, их распределение между членами команды, порядок контроля качества [9].

Методология RUP имеет ряд преимуществ:

- любые изменения или уточнения в требованиях происходят в ранний период разработки проекта, поэтому трудоемкость по их внедрению минимальна;
- она позволяет обнаружить ошибки на ранних стадиях разработки, что уменьшает вероятность возникновения рисков;
- итеративный подход способствует повторному использованию программных компонентов.

В качестве языка моделирования RUP используется Unified Modelling Language (UML) [10], который позволяет создавать диаграммы, которые моделируют бизнес-процессы и структуру разрабатываемой системы.

На основании рассмотренных достоинств технология RUP была выбрана в качестве основной технологии для разработки веб-приложения для обмена сообщениями.

Серверная часть приложения должна представлять веб-приложение, написанное на языке программирования Python 3.8 [11]. Он состоит из двух частей:

- REST API для обработки HTTP-запросов и взаимодействия с базой данных;

− асинхронная часть, использующая технологию WebSocket для пересылки между пользователями.

Для хранения данных пользователей была выбрана СУБД MongoDb – документо-ориентированная система управления базами данных, в которой данные хранятся в виде коллекций документов [12, 13].

Разработка мобильного приложения должна производится с помощью кроссплатформенной среды разработки приложений и игр Unity, оптимизированной под операционную систему Android. Поэтому для создания приложения будут использованы движок Unity 2019.5 [14], среда разработки Visual Studio 2019 и язык программирования C# 7.0 [15]. При разработке графического интерфейса будет использован редактор Paint.NET и инструмент для проектирования интерфейса Figma [16]. Достоинства выбранных технологий приведены далее.

Для тестирования серверной части будет использовано программное обеспечение для отправки HTTP-запросов Postman [17] и расширения для браузеров Simple Websocket Client и Smart Websocket Client для отправки Websocket-сообщений. Тестирование мобильного приложение будет проводиться на устройствах Meizu M6S с операционной системой Android 7.0 и Samsung A10 с операционной системой Android 10.0.

2.2 Проектирование архитектуры программного обеспечения проекта

Приложение по обмену сообщениями между пользователями предполагает обмен информацией между несколькими пользователями системы, поэтому для реализации была выбрана двухуровневая архитектура. Личные данные пользователей хранятся на сервере, предоставляющим API к ним. Также на сервере происходит обмен сообщениями между пользователями. Клиентская часть – мобильное приложение взаимодействует с API и предоставляет удобное отображение данных и возможность

отправки и получения сообщений от других пользователей. Также было принято решение, хранить историю переписки в зашифрованном виде на мобильном устройстве, с возможностью резервного сохранения в базе данных, что обеспечивает надежность сохранения данных от утечки и использования посторонними лицами.

Плюсами такой архитектуры является, что вся работа по обмену сообщениями происходит на сервере, что снижает нагрузку на устройство, на котором запущено клиентское приложение. На сервер проще организовать контроль полномочий, чтобы разрешать отправку сообщений другим пользователям. Также это позволяет избежать дублирования кода при использовании нескольких видов клиентских приложений: мобильного приложения, веб интерфейса или десктопного приложения.

2.3 Проектирование интерфейса пользователя

Пользовательский интерфейс должен быть простым и понятным в использовании для пользователей. Он должен обладать приятным дизайном обладать свойствами современных мобильных приложений. Проектирование интерфейса пользователя было проведено с помощью инструментального средства создания дизайна Figma.

На основе выделенных функциональных требований был спроектирован пользовательский интерфейс мобильного приложения. К основным страницам относятся:

- страница авторизации;
- страница регистрации;
- страница с контактами и историей переписок;
- страница переписки с конкретным пользователем;
- страница со списком контактов;
- всплывающее окно создания нового чата и добавления нового пользователя в контакты.

Рассмотрим элементы, содержащиеся на странице авторизации (рисунок 17).

Рисунок 17 – Страница авторизации

Страница авторизации состоит из следующих компонентов:

1 – поле ввода логина;

2 – поле ввода пароля;

3 – кнопка входа;

4 – кнопка перехода на страницу регистрации;

5 – сообщение об ошибке.

Данная страница содержит 2 кнопки, предназначенные для перехода к функциональным возможностям приложения. После заполнения пользователем полей ввода логина и пароля и нажатия на кнопку авторизации приложение сформирует и отправит запрос на авторизацию к серверу. В случае успешного завершения процесса откроется главная страница. По нажатию на кнопку регистрации откроется страница

регистрации. В случае возникновения ошибок в соответствующее поле (5) выведется сообщение о возникших неполадках.

Рисунок 18 – Страница регистрации

Страница регистрации состоит из следующих элементов:

1 – поле ввода логина;

2 – поле ввода пароля;

3 – поле повторения пароля;

4 – кнопка регистрации;

5 – кнопка перехода на страницу входа;

6 – сообщение об ошибке.

После заполнения полей логина, пароля и повторения пароля и нажатия на кнопку регистрации приложение сформирует и отправит запрос на регистрацию к серверу. В случае успешного завершения процесса откроется главная страница. По нажатию на кнопку входа откроется страница

авторизации. В случае возникновения ошибок в соответствующее поле (5) выведется сообщение о возникших неполадках.

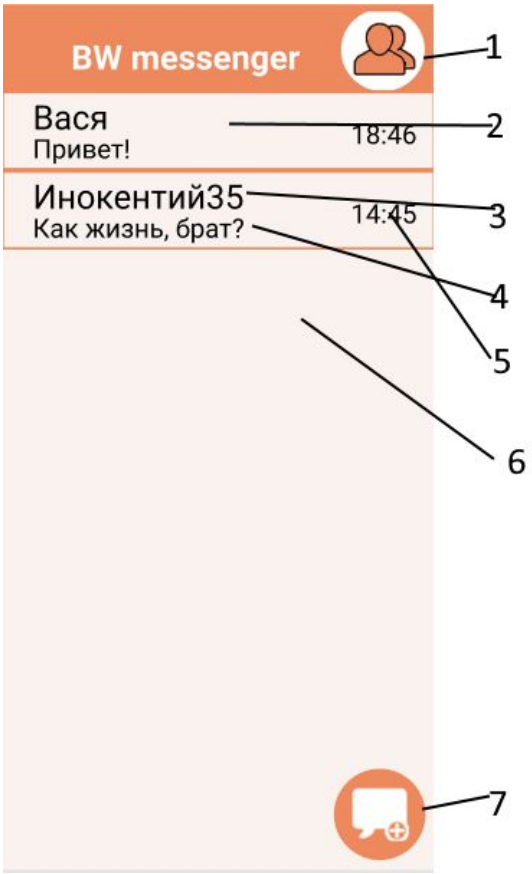

Рисунок 19 – Главная страница

Главная страница содержит следующие элементы:

1 – кнопка перехода на страницу с контактами;

2 – элемент списка контактов;

3 – имя контакта;

4 – последнее сообщение от контакта;

5 – время последнего сообщения;

6 – список контактов;

7 – кнопка перехода на всплывающее окно создания нового чата и добавления нового пользователя в контакты.

После открытия страницы с контактами и историей переписок приложение динамически заполняет список контактов с информацией о переписках пользователя. Каждая такая ячейка содержит в себе информацию

об имени пользователя, последнем сообщении в чате и его времени, по нажатии на нее открывается окно чата с выбранным контактом . По нажатии на кнопку 6 открывается окно с созданием нового чата. По нажатии на кнопку 1 открывается страница со списком контактов.

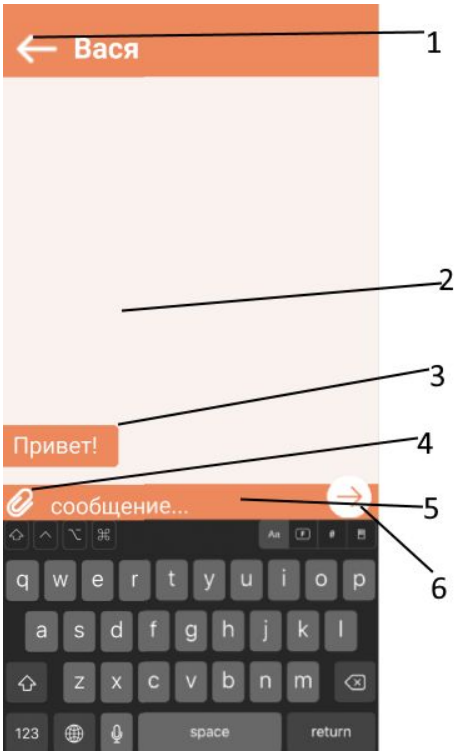

Рисунок 20 – Страница чата с конкретным пользователем

Страница чата с пользователем содержит следующие элементы:

1 – кнопка назад;

2 – контейнер для отображения всех сообщений;

3 – текстовое поле сообщения;

4 – кнопка прикрепления изображения;

5 – поле ввода сообщения;

6 – кнопка отправки сообщения.

После открытия данной страницы приложение динамически заполняет контейнер для отображения всех сообщений. После того, как пользователь введет сообщение в соответствующее поле ввода и нажмет на кнопку отправки сообщения, приложение отправит его на сервер через

Websocket-соединение. В случае получения сообщения приложение разместит его в контейнере для хранения сообщений. В случае нажатия на кнопку прикрепления изображения приложение откроет стандартное окно операционной системы Android для выбора необходимого изображения в галерее. В случае нажатия кнопки назад откроется главная страница.

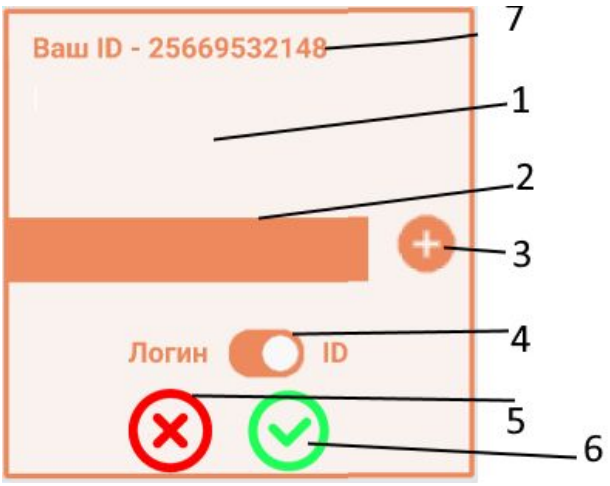

Рисунок 21 – Всплывающее окно создания нового чата и добавления нового пользователя в контакты

Всплывающее окно создания нового чата и добавления нового пользователя в контакты содержит следующие элементы:

1 – контейнер для введенных ID пользователей;

2 – поле ввода пользователя;

3 – кнопка добавления еще одного пользователя;

4 – переключатель, отвечающий за то, каким образом будет производиться поиск – с помощью логина или уникального идентификатора;

5 – кнопка закрытия окна;

6 – кнопка подтверждения;

7 – уникальный идентификатор пользователя.

Для создания нового чата или добавления нового контакта пользователю необходимо ввести логин необходимого пользователя или его уникальный идентификатор, который можно посмотреть в этом окошке.

Контролировать способ поиска можно можно с помощью соответствующего переключателя. В случае, если создается групповой чат, пользователю необходимо добавить других контактов с помощью кнопки добавления. Все введенные контакты отображаются в соответствующем контейнере. По нажатии на кнопки закрытия окна или кнопки подтверждения данное окно закрывается, и открывается главная страница. Но в последнем случае приложение отправляет запрос на сервер о создании нового чата.

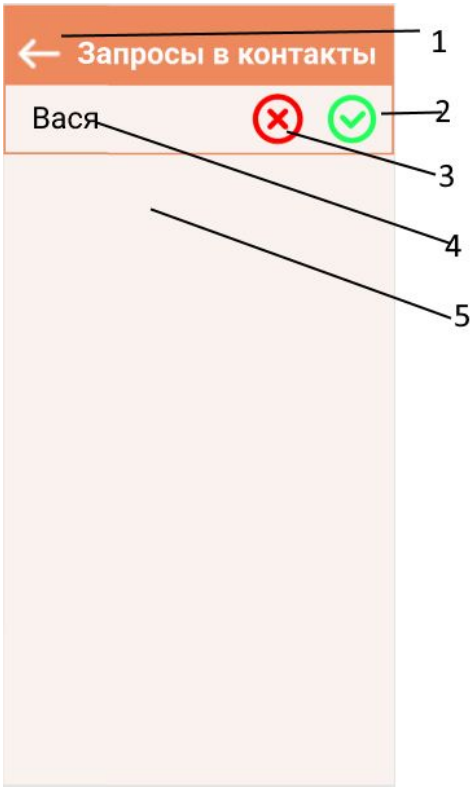

Рисунок 22 – Страница со списком контактов

Страница со списком контактов содержит следующие элементы:

1 – кнопка перехода на главную страницу;

2 – кнопка принятия приглашения на добавление контактов;

3 – кнопка отказа приглашения на добавление контактов;

4 – имя контакта;

5 – контейнер для списка контактов;

После открытия страница в контейнер для контактов динамически добавляются запросы на добавления пользователя в контакты. По нажатии на

кнопки принятия или отказа приглашения на сервер отправится запрос с ответом пользователя, а сама запись из контейнера удалится. По нажатии на кнопку назад произойдет переход на главный экран.

2.4 Проектирование серверной части приложения

Для хранения данных была выбрана MongoDb – документо-ориентированная СУБД с открытым исходным кодом. MongoDb не использует традиционный реляционный способ представления данных. Данная СУБД основана на принципе хранения документов в формате BSON (binary JSON). Каждая запись представляет собой документ без заранее заданной схемы. Такой формат хранения данных имеет следующие преимущества:

- интуитивно понятный способ отображения данных;
- динамические запросы;
- эффективное хранение двоичных данных;
- гибкий язык для формирования запросов;
- поддержка индексов.

Также она обладает надежностью и эффективностью, поддерживается всеми популярными языками программирования и позволяет отобразить отношения предметной области.

Для серверной части был выбран язык программирования Python. Python – универсальный высокоуровневый язык программирования. Он имеет очень широкую сферу применения – начиная от разработки Web-приложений, заканчивая построением систем искусственного интеллекта. К преимуществам Python относятся:

- динамическая типизация;
- интерпретируемость;
- хорошая поддержка модульности;
- автоматический сборщик мусора;

- понятный синтаксис;
- большое количество написанных библиотек и фреймворков, позволяющих решить большое количество задач;
- кроссплатформенность.

Исходя из этого, язык программирования Python позволит решить все поставленные задачи с наименьшими затратами.

В качестве среды разработки серверной части web-приложения использовалась бесплатная версия программного продукта PyCharm компании JetBrains.

2.4.1 Проектирование базы данных

База данных MongoDb является документо-ориентированной СУБД, в которой данные хранятся в виде коллекций документов. В отличие от привычных всем реляционных баз данных, где данные хранятся в виде таблиц, в MongoDb данные представлены в виде набора коллекций, внутри которых данные хранятся в BSON формате. Большим достоинством такого подхода является гибкость представления данных, это означает, что внутри коллекции нет четко выделенной структуры, и она может изменяться от документа к документу.

На основе анализа функциональных требований была спроектирована база данных. Она состоит из двух коллекций:

- chat_db, в которой хранятся личные данные пользователей;
- group_chats_db, в которой хранятся данные о групповых чатах пользователей.

На рисунке 23 изображена структура документа коллекции chat_db.

```
{
    _id,
    login,
    password,
    contacts: [],
    contacts_requested: [],
    not_confirmed_contacts: [],
    unsended_messages: {},
    group_chats_ids: []
}
```

Рисунок 23 – Структура документа коллекции chat_db

В составе документа коллекции chat_db следующие элементы:

- _id – уникальный идентификатор объекта, генерируется СУБД;
- login – логин пользователя;
- password – зашифрованный пароль пользователя;
- contacts – список контактов данного пользователя;
- contacts_requested – список пользователей, которые отправили запрос данному пользователю;
- not_confirmed_contacts – список пользователей, которым данный пользователь отправил запрос;
- unsended_messages – словарь, хранящий в себе неоправленные данному пользователю сообщения. Такое произойдет в случае, если данный пользователь не был в приложении, и в это время ему отправит сообщение другой пользователь;
- group_chats_ids – список уникальных идентификаторов групповых чатов, в которых состоит пользователь.

На рисунке 24 приведена структура документа коллекции group_chats_db.

```
{
    _id,
    name,
    members: [],
    unsended_messages: {}
}
```

Рисунок 24 – Структура документа коллекции group_chats_db

В составе документа коллекции group_chats_db имеются следующие элементы:

- _id – уникальный идентификатор объекта, генерируется СУБД;
- name – название чата;
- members – список, содержащий уникальные идентификаторы пользователей, состоящих в данном чате;
- unsended_messages – словарь, хранящий в себе неоправленные сообщения.

Проектирование базы данных позволяет отобразить модель предметной области в объекты, используемые сервером.

2.4.2 Проектирование API

Серверная часть приложения представляет состоит из двух частей:

- REST интерфейс, позволяющий использовать стандартные методы HTTP (GET, POST) и использующий для передачи данных формат JSON [18];
- асинхронная часть, работающая на технологии Websocket, позволяющая передавать сообщения в режиме реального времени [19].

В языке программирования Python для разбиения кода на логические составляющие используется механизм модулей. Выделим в приложении следующие модули:

- common_types;
- database;
- handlers;
- messages;
- services.

Модуль handlers содержит в себе обработчики, сопоставляющиеся каждому URL адресу, наследуемые от tornado.web.RequestHandler. Список

всех обработчиков, необходимых для покрытия всех функциональных требований:

- add_contact_by_login;
- add_contact_handler;
- confirm_contact_handler;
- get_all_contacts_handler;
- login_handler;
- register_handler;
- create_multichat_handler;

Каждому обработчику соответствует класс из модуля messages, который содержит в себе структуру сообщения в формате JSON для клиент-серверного взаимодействия.

Асинхронная часть, работающая на технологии Websocket, представляет собой класс chat_handler, наследуемый от tornado.websocket.WebSocketHandler. Он является обработчиком событий, связанных с открытием соединения с клиентом, получения и отправки сообщений.

Модуль services содержит классы, предоставляющие методы работы с пользователями, отправкой сообщений и другой необходимой логикой приложения. Они используют модуль database для взаимодействия с базой данных.

3 Реализация веб-приложения

3.1 Реализация серверной части приложения

3.1.1 Реализация базы данных

Развертывание базы данных было выполнено запуском скрипта, встроенного в СУБД MongoDb. Он отслеживает текущую версию данных, позволяет вносить новые изменения и делать откат до предыдущих версий.

Исходный код создания базы данных представлен в листинге 1.

Листинг 1 – Скрипт создания базы данных

```
use chat_db()
db.chat_db.insert_one({“test”: “test»})
db.group_chats_db.insert_one({“test”: “test»})
```

База данных была развернута на локальном сервере Mongo.

3.1.2 Разработка серверной части приложения

Для разработки веб-приложения был использован язык программирования Python и его фреймворк Tornado [20, 21], позволяющий реализовать и REST API, и канал связи Websocket.

При старте серверного программного обеспечения запускается инструмент работы с базами данных, который проверяет, что она была развернута и имеется подключение. В случае возникновения ошибок сработает код обработки возникшей исключительной ситуации, чтобы предупредить разработчика об этом.

Были реализованы обработчики всех спроектированных запросов. Для всех ресурсов кроме авторизации и регистрации требуется HTTP заголовок jwt, содержащий в себе уникальный токен пользователя. Серверная часть проверяет, соответствует ли данный идентификатор авторизованному пользователю, в случае, если данные неверны, то прерывает выполнения обработчика и возвращает код статуса 403 – доступ запрещен.

Обработчики запросов выполняют проверку, авторизован ли пользователь, далее выполняет конвертацию данных из строки в JSON формате в классы приложения и вызывают методы модуля services для обработки данных, который работает с модулем database, являющийся оберткой над базой данных, позволяющий абстрагироваться от выполнения запросов.

Основная бизнес логика работы приложения реализована в модуле services. Обработка запросов на добавление пользователя в контакты, создание групповых чатов обрабатываются в классе SocialService, который получает необходимые данные из базы данных от модуля database, формирует ответ на запрос и возвращает в обработчики запросов. Также, добавляет необходимые записи в базу данных о подтверждении запроса добавления контактов. Работа с пользователями, находящимися в приложении происходит в классе OnlineUsersService. В случае, если первый пользователь отправляет сообщение второму, этот класс отвечает за то, возможно ли доставить сообщение второму в данный момент через Websocket соединение или сделать запись в базы данных до тех пор, когда пользователь появится в сети.

Класс WebsocketService отвечает за открытие, поддержание и последующее закрытие между пользователем и сервером. В отличие от REST API, Websocket соединение между клиентом и сервером устанавливается 1 раз по заданному URL, после чего по этому соединению передаются сообщения в формате JSON. Для различения разных типов запросов, в модели данных каждого сообщения было добавлено поле type.

Для обеспечение безопасности данных на сервер было добавлено шифрование и дешифрование данных, заносимых в базу данных, реализованное в классе EncryptionService, который использует алгоритм шифрования RSA [22]. Такой подход используется при регистрации, при которой пароль в зашифрованном виде заносится в базу данных.

Полный исходный код серверной части представлен в Приложении Б.

3.2 Реализация мобильного приложения

Разработка приложения производилась с помощью кроссплатформенной среды разработки Unity с использованием языка программирования C# [23, 24]. Полный исходный код приложения представлен в Приложении В. На рисунках 25, 26 представлена диаграмма классов данного приложения.

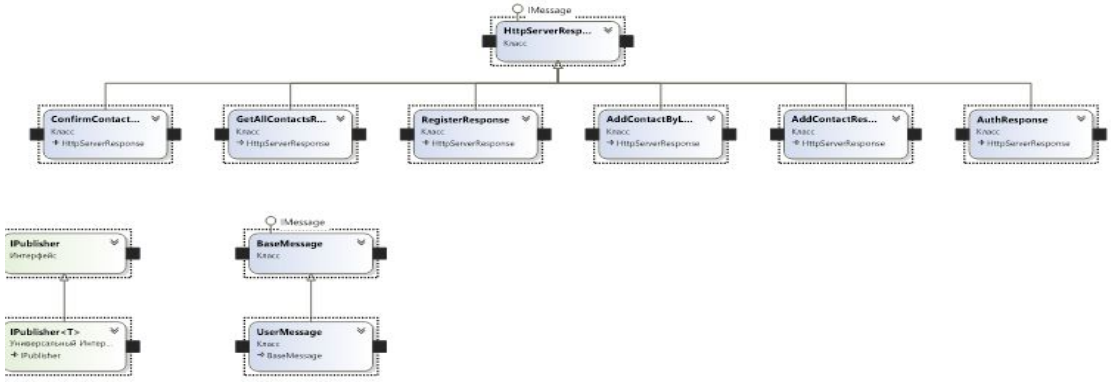

Рисунок 25 – Диаграмма классов приложения

Рисунок 26 – Продолжение диаграммы классов приложения

При разработке приложения применялась методология разработки Dependency Injection (DI), сутью которой является то, что поля объектов задаются внешней сущностью, что увеличивает гибкость кода и возможность его повторного использования без значительных доработок.

Ниже приведена спецификация основных классов приложения:

- AppManager – точка входа в приложение и его главный DI-контейнер, в котором создаются основные сущности и происходит установление связей между ними;
- NetworkManager – главный менеджер для взаимодействия с сервером, отправляет и обрабатывает запросы пользователя;
- EncryptionService – отвечает за шифрование и дешифрование данных, отправляемых по сети и сохраняемых на устройстве;
- FileManager – класс, отвечающий за сохранение, добавление и извлечение данных, хранимых на устройстве, это применяется к истории сообщений пользователя, которая шифруется алгоритмом RSA, для сохранения на устройстве используется механизм PlayerPrefs – встроенный в Unity механизм сохранения данных на устройстве в формате ключ-значение;
- MessagesManager – менеджер, отвечающий за получение и сохранение истории сообщений пользователя;
- HttpNetworkService – сервис, отвечающий за отправку и получение HTTP запросов к серверу;
- Session – класс-хранилище персональных данных пользователя;
- JsonDecoder – сериализует и десериализует данные в JSON формат;
- WebSocketAdapter – оболочка над встроенным классом Websocket;
- WebSocketService – сервис, отвечающий за отправку и получение сообщений через Websocket соединение;

− PageManager – менеджер, контролирующий навигацию по страницам приложения;

− MainPageController – контроллер главного экрана приложения;

− LoginPageController – контроллер экрана авторизации;

− RegisterPageController – контроллер экрана регистрации;

− DialogPageController – контроллер экрана диалога;

− AddContactPageController – контроллер экрана добавления нового пользователя в контакты;

− RequestedContactsPageController – контроллер экрана контактов;

− NetworkConfig – конфиг настроек соединения с сервером;

− SecureConfig – конфиг настроек шифрования.

Также помимо основных уже описанных менеджеров и сервисов в приложении присутствует набор классов, отвечающих за представление моделей сообщений и моделей данных.

Исходный код мобильного приложения представлен в приложении В.

Диаграмма развертывания web-приложения представлена на рисунке 27. Она предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее выполнения.

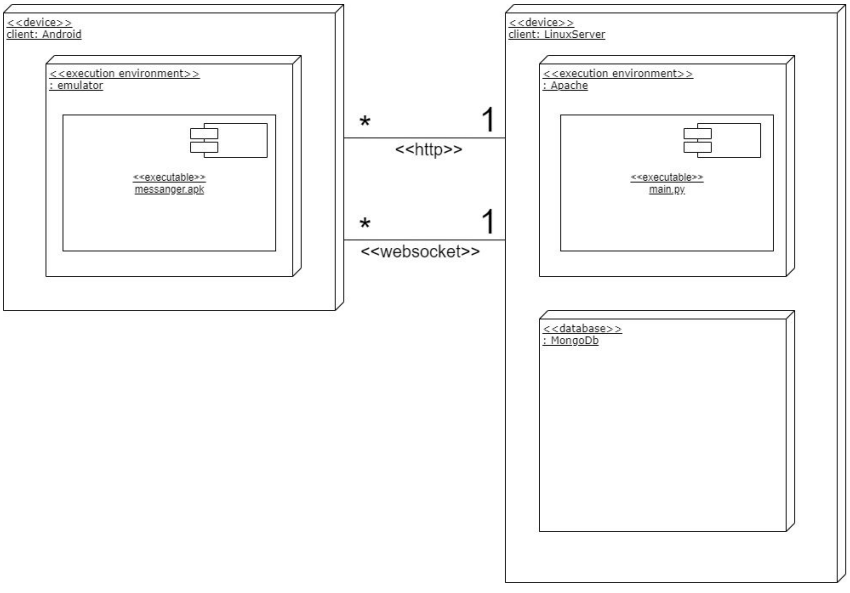

Рисунок 27 – Диаграмма развертывания web-приложения

Серверная часть приложения разворачивается на устройстве, работающем на Unix-подобной системе. Также на нем запускается сервер базы данных. Клиентское приложение разворачивается на мобильных устройствах под управлением операционной системы Android и взаимодействует с сервером с помощью протоколов HTTP и Websocket.

На рисунке 28 приведена диаграмма компонентов, отражающая общую структуру исходного кода программы.

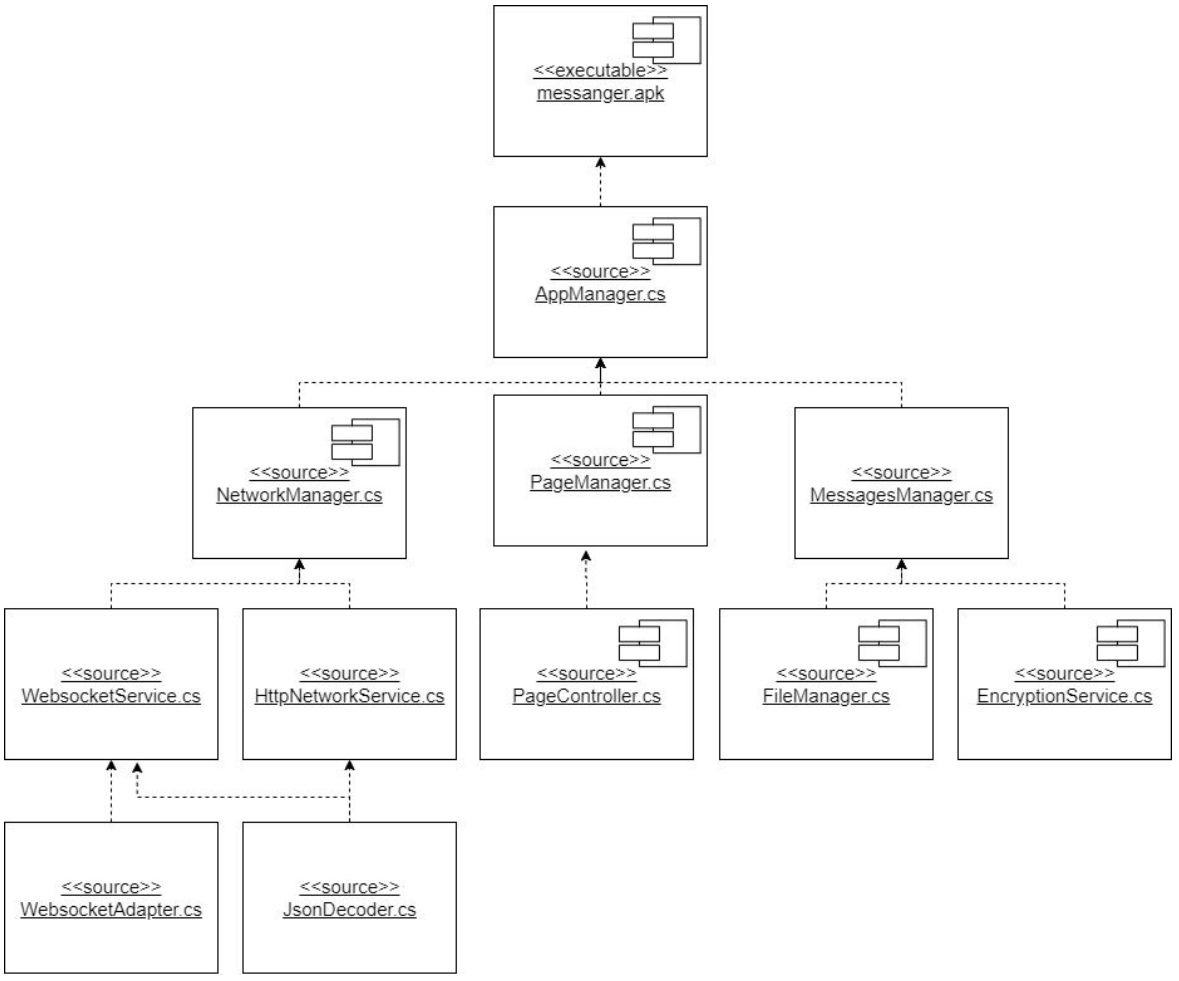

Рисунок 28 – Диаграмма компонентов мобильного приложения

Компонентами данного приложения являются файлы, содержащие исходный код класса. В итоге все компоненты становятся частью главного класса AppManager, в котором идет их инициализация.

4. Контроль качества программного обеспечения

4.1 Тестирование серверной части

Для того чтобы убедиться, что разрабатываемая серверная часть корректно работает, необходимо протестировать каждый из запросов к API. Тестирование HTTP запросов было выполнено с помощью инструментального средства Postman, тестирование работы Websocket-соединения было выполнены с помощью плагинов для браузера Chrome SimpleWebsocketClient и SmartWebsocketClient.

Рассмотрим выполнение HTTP запросов. Отправляем POST запрос на регистрацию нового пользователя с обязательными полями логина и пароля. При успешном прохождении регистрации сервер вернет пользователю сообщение об успехе и уникальный JWT токен. Получен результат с кодом ответа 200 [25], означающий, что операция выполнена (рисунок Г.1). При попытке зарегистрироваться с логином, который уже есть в базе данных, сервер вернет код ответа 409 – конфликтная ситуация (рисунок Г.2). При отсутствии какого-нибудь поля запроса или его некорректным отображением будет получен код ответа 400 – неверные параметры (рисунок Г.3).

Рассмотрим GET запрос на получение списка всех контактов пользователя. Для успешного выполнения этого запроса необходимо перед этим получить JWT токен путем прохождения авторизации. Если попытаться отправить запрос, с неправильным параметром токена, то сервер вернет код ответа 400 – неверные параметры (рисунок Г.4). В случае успешного выполнения запроса сервер вернет код ответа 200 и объект JSON-формата, содержащий в себе 3 списка – контакты, запрос в контакты и неподтвержденные контакты (рисунок Г.5).

Аналогичным образом было проведено тестирование оставшихся ресурсов API серверного приложения Полные результаты тестирования всех запросов представлены в таблице 7.

Таблица 7 – План и результаты тестирования REST API

Номер	Описание теста	Запрос		Результат
		Метод	URI	
1	Авторизация пользователя без параметров	POST	/login	Тест пройден. Неверные параметры запроса. Код ответа - 400
2	Авторизация пользователя с неверным паролем	POST	/login	Тест пройден. Код ответа - 403
3	Авторизация пользователя с корректными данными	POST	/login	Тест пройден. Код ответа - 200. Получен JWT токен
4	Регистрация пользователя с неверными данными	POST	/register	Тест пройден. Код ответа - 400
5	Регистрация пользователя с уже зарегистрированным логином	POST	/register	Тест пройден. Код ответа - 409
6	Регистрация пользователя с корректными данными	POST	/register	Тест пройден. Код ответа - 200
7	Добавить пользователя в контакты с несуществующим ID	POST	/add_contact	Тест пройден. Код ответа - 400
8	Добавить пользователя в контакты	POST	/add_contact	Тест пройден. Код ответа - 200
9	Добавить пользователя в контакты с несуществующим ID	POST	/add_contect_by_login	Тест пройден. Код ответа - 400
10	Добавить пользователя в контакты с помощью логина	POST	/add_contect_by_login	Тест пройден. Код ответа - 200
11	Получение списка контактов при неверно указанном токене	GET	/get_all_contacts	Тест пройден. Код ответа - 400
12	Получение списка контактов	GET	/get_all_contacts	Тест пройден. Код ответа - 200
13	Подтвердить контакт с неверно указанным параметром	POST	/confirm_contact	Тест пройден. Код ответа - 400
14	Подтвердить контакт	POST	/confirm_contact	Тест пройден. Код ответа - 200
15	Создание группового чата без списка участников	POST	/create_multichat_handler	Тест пройден. Код ответа - 400
16	Создание группового чата	POST	/create_multichat_handler	Тест пройден. Код ответа - 200

Рассмотрим отправку сообщений через Websocket соединение. Для этого необходимо авторизоваться двумя аккаунтами и получить JWT токены с помощью Postman. С помощью плагинов для браузера Simple Websocket Client и Smart Websocket Client протестируем отправку сообщений. Отправим сообщение от первого клиента второму (рисунок Г.6). Сообщение успешно дошло до второго клиента (рисунок Г.7).

Аналогичным образом было проведено тестирование остальных типов сообщений по протоколу Websocket. Результаты тестирования приведены в таблице 8.

Таблица 8 – План и результаты тестирования Websocket

Номер теста	Описание теста	Тип сообщения	Результат
1	Отправка обычного сообщения без указания JWT токена	to_server	Тест пройден. Пользователю вернулось сообщение об ошибке
2	Отправка обычного сообщения без указания получателя	to_server	Тест пройден. Пользователю вернулось сообщение об ошибке
3	Отправка обычного сообщения	to_server	Тест пройден. Сообщение доставлено получателю
4	Отправка изображения без указания JWT токена	image	Тест пройден. Пользователю вернулось сообщение об ошибке
5	Отправка изображения без без указания получателя	image	Тест пройден. Пользователю вернулось сообщение об ошибке
6	Отправка изображения	image	Тест пройден. Сообщение доставлено получателю
7	Отправка сообщения в групповой чат без указания JWT токена	to_server	Тест пройден. Пользователю вернулось сообщение об ошибке
8	Отправка сообщения в групповой чат без указания ID чата	to_server	Тест пройден. Пользователю вернулось сообщение об ошибке
9	Отправка сообщения в групповой чат	to_server	Тест пройден. Сообщение доставлено получателю

Все тесты API и соединения Websocket были успешно пройдены, что свидетельствует об их корректной работе.

4.2 Тестирование клиентской части

4.2.1 Функциональное тестирование

Тестирование программного обеспечения – исследовательский процесс, который имеет цель проверить соответствие между ожидаемым и реальным поведением программы.

Необходимо провалидировать данные при регистрации нового пользователя, а конкретно его пароль. Он должен состоять из латинских символов, иметь только допустимые символы и быть длиннее 6 символов. Только в этом случае пользователь сможет создать новую учетную запись.

4.2.2 Метод эквивалентного разбиения

В таблице 9 приведены классы эквивалентности для поля пароля.

Таблица 9 – Классы эквивалентности

Входные данные	Класс эквивалентности	Выходные данные	Полученный результат
-	Пустое значение	Невозможно зарегистрироваться	Невозможно зарегистрироваться (рисунок Д.1)
мой_пароль	Присутствует кириллица	Невозможно зарегистрироваться	Невозможно зарегистрироваться (рисунок Д.2)
password%	Присутствует недопустимый символ %	Невозможно зарегистрироваться	Невозможно зарегистрироваться (рисунок Д.3)
hi	Значение длиной менее 7 символов	Невозможно зарегистрироваться	Невозможно зарегистрироваться (рисунок Д.4)
strong_password44	Строка, состоящая из латиницы, цифр, длиной более 7 символов	Регистрация пройдена	Регистрация пройдена (рисунок Д.5)
strong_password44 и strong_password43	Пароли не совпадают	Невозможно зарегистрироваться	Невозможно зарегистрироваться (рисунок Д.6)

Все тесты успешно пройдены, полученные результаты соответствуют ожидаемым.

4.3 Метрики кода

Приведенный в листинге 2 метод предназначен для отправки полученного от первого пользователя сообщения второму. В случае, если он находится в приложении, метод находит Websocket соединение с ним и отправляет сообщение, в противном случае, сохраняет данные в базу данных до тех пор, пока второй пользователь не зайдет в приложение.

Листинг 2 – Метод отправки сообщения

```
def message_to_server_handler(self, message: str):
    deserialized_message = SendMessageToServer.deserialize(message)
    receiver = deserialized_message.receiver
    # Receiver онлайн
    if self._online_user_service.is_user_online
    (deserialized_message.receiver):
       receiver_socket = self._online_user_service.get_socket(
           deserialized_message.receiver)
       receiver_socket.write_message(
           '{"type": "to_client", "message": %s, "sender": %s}' %
           (deserialized_message.message,
            self._online_user_service.get_user_id
           (deserialized_message.jwt)))
    # Receiver not online
    else:
       self._database_service.add_unsended_message(
       receiver,
       self._online_user_service.get_user_id(
       deserialized_message.jwt),
       deserialized_message.message)
```

Далее рассчитаны метрики кода для метода отправки сообщения.

4.3.1 Метрики Холстеда

Метрики Холстеда являются показателями качества кода программного обеспечения. Входными данными для метрик Холстеда [26] являются уникальные операторы и операнды, представленные в таблицах 10, 11

Таблица 10 – Уникальные операторы

Номер	Оператор
1	=
2	if
3	else
4	%
5	SendMessageToServer.deserialize
6	_online_user_service.is_user_online
7	_online_user_service.get_socket
8	write_message
9	_online_user_service.get_user_id
10	_database_service.add_unsended_message

Таблица 11 – Уникальные операнды

Номер	Операнды
1	deserialized_message
2	receiver
3	receiver_socket
4	message
5	jwt

Метрики Холстеда основаны на следующих параметрах:

- n1 – число уникальных операторов программы;
- n2 – число уникальных операндов программы;
- N1 – общее число операторов в программе;
- N2 – общее число операндов в программе.

Учитывая обозначения, можно определить словарь программы:

$$n = n1 + n2.$$

Длина программы:

$$N = N1 + N2.$$

Теоретическая длина программы:

$$N' = n1 * \log_2 n1 + n2 * \log_2 n2.$$

Объем программы:

$$V = N * \log_2 n.$$

Уровень качества программирования без учета теоретических параметров:

$$L' = (2 * n2) / (n1 * N2).$$

Сложность понимания программы:

$$E_c = V / (L' * 2).$$

Результаты расчетов представлены в таблице 12.

Таблица 12 – Расчет метрик Холстеда

Метрика	Значение
n1	10
n2	5
N1	13
N2	9
n	15
N	22
V	3.9
N’	118
L’	0.08
E_c	24

Сложность понимания программы менее 75, что показывает, что разработанный код имеет высокий уровень качества, что позволяет без трудностей поддерживать, изменять и дорабатывать его при поддержке проекта.

4.3.2 Цикломатическая метрика Мак-Кейба

Показатель МакКейба отображает цикломатическую сложность программного кода – счетное число линейно независимых маршрутов через программный код. Цикломатическая сложность является одной из метрик программного обеспечения, которая позволяет количественно оценить логическую сложность программы [27].

При вычислении цикломатической сложности используется граф потока управления программы. Узлы графа соответствуют неделимым группам команд программы, они соединены ориентированными рёбрами, если группа команд, соответствующая второму узлу, может быть выполнена непосредственно после группы команд первого узла. Она определяется по формуле:

$$V(G) = e - n + 2 * p,$$

где e – количество дуг;

n – количество вершин:

p – число компонент связности.

Расчет метрики Мак-Кейба представлен в таблице 13.

Таблица 13 – Расчет метрики Мак-Кейба

Метрика	Значение
e	5
n	5
p	1
V(G)	2

Полученное значение цикломатической метрики МакКейба меньше 10. Это показывает, что цикломатическая сложность программы находится на низком уровне.

4.4 Модульное тестирование

Модульное тестирование (unit testing) – вид тестирования, задача которого проверить каждый модуль системы по отдельности. Units – модульные тесты, применяемые в различных слоях приложения, тестирующие наименьшую логику приложения: например, класс, но чаще всего метод. Эти тесты обычно стараются по максимуму изолировать от внешней логики, то есть создать иллюзию того, что остальная часть приложения работает в стандартном режиме

Приведенный в листинге 3 метод предназначен для авторизации пользователя в системе. В случае его успешного выполнения он вернет словарь, содержащий в себе результат выполнения и JWT строку. В противном случае возвращаемый словарь будет содержать ошибку. Некорректное поведение могут вызвать неправильно введенные данные и ошибки соединения. Данный тест должен проверить оба случая: корректные и некорректные входные данные. Тестирование фрагмента кода проведено с помощью стандартной библиотеки языка программирования Python unittest [28]. Код тестирования приведен в листинге 4.

Листинг 3 – Метод авторизации

```
def login(self, message: LoginMessage):
        db_result, id = self._database_service.login(message.login,
    message.password)
       if db_result == LoginResults.success:
           jwt = jwt_service.create_jwt(id)
           response = dict({"result": "success", "jwt": str(jwt)})
           self._online_users_service.add_user_online(jwt, id)
       elif db_result == LoginResults.error or db_result ==
    LoginResults.not_validated or db_result ==
    LoginResults.server_error:
           response = dict({"result": "failed"})
       return response
```

Листинг 4 – Код unit теста

```
class Test(unittest.TestCase):
```

```
    def test_login_correct(self):
        client = DatabaseService()
        online_users_service = OnlineUsersService()
        login_service = LoginService(client, online_users_service)
        result = login_service.login(LoginMessage("dima2", "123"))
        self.assertEqual(result["result"], "success")

    def test_login_wrong(self):
        client = DatabaseService()
        online_users_service = OnlineUsersService()
        login_service = LoginService(client, online_users_service)
        result = login_service.login(LoginMessage("dima3", "123"))
        self.assertEqual(result["result"], "failed")
```

Результат работы тестов представлен на рисунке 29. Unit тесты были успешно пройдены.

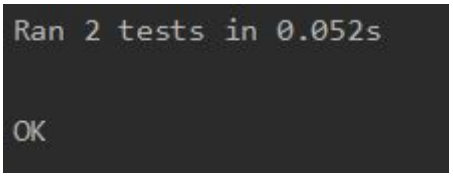

Рисунок 29 – Результаты unit тесты метода login

Приведенный в листинге 5 метод проверяет, находится ли нужный пользователь в приложении. Для этого он проходит по всем подключениям, в случае, если находится необходимый пользователь, то метод вернет логическое значение True, в противном случае, возвращается значение False. Данный тест должен проверить оба случая: пользователь находится в приложении и пользователь не находится в приложении. Код тестирования приведен в листинге 6.

Листинг 5 – Метод is_user_online()

```
def is_user_online(self, id):
    for v in self.__users_online.values():
        if v.id == id:
            return True
    return False
```

Листинг 6 – Код unit теста

```
def test_use_online_correct(self):
    client = DatabaseService()
```

```
    online_users_service = OnlineUsersService()
    login_service = LoginService(client, online_users_service)
    login_service.login(LoginMessage("dima2", "123"))

self.assertEqual(online_users_service.is_user_online("559872236547895
3"), True)

def test_use_online_wrong(self):
    client = DatabaseService()
    online_users_service = OnlineUsersService()
    login_service = LoginService(client, online_users_service)
    login_service.login(LoginMessage("dima2", "123"))

self.assertEqual(online_users_service.is_user_online("559872246457789
53"), False)
```

Результат работы тестов представлен на рисунке 30. Unit тесты были успешно пройдены.

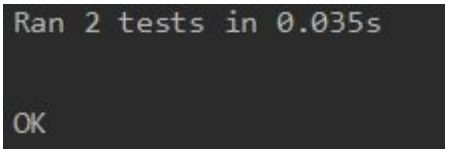

Рисунок 30 – Результаты unit тесты метода is_user_online

Все unit тесты были успешно пройдены, что свидетельствует о корректности работы программы.

В рамках данного раздела было произведено тестирование различных методов программного обеспечения, что позволяет сделать вывод о том, что программа работает корректно при различном наборе вводимых данных и в самых различных условиях. Был проведен анализ кода на основе метрик Холстеда и МакКейба, который позволил сделать вывод о качестве кода, его читаемости и объеме.

Заключение

В результате выполнения выпускной квалификационной работы было разработано серверное программное обеспечение и мобильное приложение, позволяющее пользователям информационной системы обмениваться сообщениями между собой.

На этапе анализа предметной области выявлены функциональные и нефункциональные требования к приложению и рассмотрены его аналоги на рынке систем обмена сообщениями.

На этапе проектирования был составлен план разработки программного обеспечения, сформирована последовательность работ, рассчитан бюджет проекта и его продолжительность. Для реализации необходимо 667200 руб., разработка займет 59 дней, что говорит о целесообразности разработки приложения.

Серверное программное обеспечение было разработано с помощью языка программирования Python в среде PyCharm. Мобильное приложение реализовано с помощью среды разработки Unity на языке программирования C#. Оно содержит 6 экранов и предоставляет пользователю весь необходимый интерфейс для использования всех функций web-приложения.

Был проведен контроль качества разработанного программного обеспечения. Рассчитанные метрики кода показали достаточно хорошее качество кода. Корректность и отказоустойчивость проверены с помощью функционального, структурного и модульного видов тестирования.

В дальнейшем планируется развернуть серверную часть на облачном сервисе Heroku и опубликовать мобильное приложение в магазине приложения для операционной системы Android GooglePlay. Таким образом, скачать приложение и воспользоваться им сможет любой пользователь, имеющий смартфон с операционной системой Android 7.0 и выше.

Список использованных источников

1. Программа «Цифровая экономика Российской Федерации». Утверждена распоряжением Правительства Российской Федерации от 28 июля 2017 г. № 1632-р. [Электронный ресурс]. – URL: http://static.government.ru/media/files/9gFM4FHj4PsB79I5v7yLVuPgu4bvR7M0.pdf (дата обращения: 19.05.2020).

2. Распоряжение Президента Российской Федерации от 9 мая 2017 г. № 203 «Стратегия развития информационного общества в Российской Федерации на 2017 – 2030 годы» [Электронный ресурс]. – URL: http://zmedu.hostedu.ru/files/ykaz_7668.pdf (дата обращения: 19.05.2020).

3. Рынок мобильных приложений: развитие и перспективы // Стандарты и качество [Электронный ресурс]. – URL: https://ria-stk.ru/ds/adetail.php?ID=166453 (дата обращения: 19.05.2020).

4. Статистика самых популярных мессенджеров на октябрь 2019 года // Statista [Электронный ресурс]. – URL: https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/ (дата обращения: 19.05.2020).

5. Эпоха мессенджеров. Какие из них самые популярные? // Sostav [Электронный ресурс]. – URL: https://www.sostav.ru/publication/kak-vladeltsy-smartfonov-ispolzuyut-messendzhery-30288.html (дата обращения: 19.05.2020).

6. Попов Ю. И. Управление проектами : учебное пособие / Ю. И. Попов, О. В. Яковенко. – М. : ИНФРА-М, 2007. – 208 с.

7. Кудрявцев Е.М. Методы сетевого планирования и управления проектом / Е.М. Кудрявцев. – М.: Книга по Требованию, 2009. – 238 с.

8. Скороход С.В.Управление проектами средствами Microsoft Project. Курс лекций. Издательство: Интуит НОУ – 2016. – 319 с.

9. RUP. Общие сведения // Informicus [Электронный ресурс]. – URL: http://www.informicus.ru/default.aspx?SECTION=6&id=73&subdivisionid=7 (дата обращения: 23.05.2020).

10. Язык uml, примеры, uml диаграммы // gameinstitute [Электронный ресурс]. – URL: http://gameinstitute.ru/uroki-uml/ (дата обращения: 18.05.2020).

11. Руководство по языку Python // Metanit [Электронный ресурс]. – URL: https://metanit.com/python/tutorial/ (дата обращения: 19.05.2020).

12. Введение в MongoDb // Metanit [Электронный ресурс]. – URL: https://metanit.com/nosql/mongodb/1.1.php (дата обращения: 19.05.2020).

13. MongoDb in python // MongoApi [Электронный ресурс]. – URL: https://api.mongodb.com/python/current/tutorial.html (дата обращения: 19.05.2020) .

14. C# и .Net. Введение // Metanit [Электронный ресурс]. – URL: https://metanit.com/sharp/tutorial/1.1.php (дата обращения: 26.05.2020).

15. Unity Manual // Unity Docs [Электронный ресурс]. – URL: https://docs.unity3d.com/Manual/index.html (дата обращения: 30.05.2020).

16. Веб дизайн в Figma с нуля и не только // Figma [Электронный ресурс]. – URL: http://figmadesign.ru/ (дата обращения: 19.05.2020).

17. Postman – ваш помощник в тестировании API // software-testing [Электронный ресурс]. – URL:https://software-testing.ru/library/testing/testing-tools/2638-postman (дата обращения: 20.05.2020).

18. Введение в Rest API // habr [Электронный ресурс]. – URL: https://habr.com/ru/post/483202/ (дата обращения: 16.05.2020).

19. Асинхронный веб // TProger [Электронный ресурс]. – URL: https://tproger.ru/translations/what-are-web-sockets (дата обращения: 23.05.2020)

20. Введение в Tornado // Habr [Электронный ресурс]. – URL: https://habr.com/ru/sandbox/27705/(дата обращения: 19.05.2020)

21. Учебник Tornado // TornadoWeb [Электронный ресурс]. – URL: https://www.tornadoweb.org/en/stable/ (дата обращения: 19.05.2020)

22. MD5 // Wikipedia [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/MD5 (дата обращения: 25.05.2020).

23. Комплексное руководство по дизайну мобильных приложений // uxpub [Электронный ресурс]. – URL: https://ux.pub/kompleksnoe-rukovodstvo-po-dizaynu-mobilnyh-prilozheniy/ (дата обращения: 15.05.2020).

24. Unity UI // Unity Docs [Электронный ресурс]. – URL: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/index.html (дата обращения: 29.05.2020).

25. Коды ответа HTTP // MDN web tools [Электронный ресурс]. – URL:https://developer.mozilla.org/ru/docs/Web/HTTP/Status (дата обращения: 20.05.2020).

26. Метрики Холстеда // Studfiles [Электронный ресурс]. – URL: https://studfile.net/preview/1568115/page:2/ (дата обращения: 21.05.2020).

27. Метрика МакКейба // Studfiles [Электронный ресурс]. – URL: https://studfile.net/preview/6833163/page:3/ (дата обращения: 23.05.2020).

28. Модуль unittest: тестируем свои программы // pythonworld [Электронный ресурс]. – URL: https://pythonworld.ru/moduli/modul-unittest.html (дата обращения: 25.05.2020).

Приложение А

ГЛОССАРИЙ

(справочное)

Мессенджер – система обмена мгновенными сообщениями в режиме реального времени.

Плагин – независимо компилируемый программный модуль, подключаемый к основной программе.

Токен – ключ доступа при запросах к API.

Фреймворк – программная платформа, определяющая структуру программной системы.

Android – мобильная операционная система для смартфонов, планшетов и т.д., разрабатываемая компанией Google.

BSON – бинарная форма представления структур данных и ассоциативных массивов.

JSON – текстовый формат обмена данных, основанный на ассоциативных массивах.

JWT – открытый стандарт для создания уникальных токенов доступа.

RUP (Rational Unified Process) – методология разработки программного обеспечения, которая использует итеративный подход к разработке.

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня передачи данных, основанный на модели «клиент-сервер»

UML (Unified Modeling Language) – язык графического описания объектного моделирования для разработки программного обеспечения, моделирования бизнес-процессов, отображения структур и системного проектирования.

Приложение Б

ИСХОДНЫЙ КОД СЕРВЕРНОЙ ЧАСТИ

(рекомендуемое)

Листинг Б.1 - Файл main.py

```
import asyncio

import tornado.httpserver
import tornado.web
import tornado.ioloop

from handlers.login_handler import LoginHandler
from handlers.register_handler import RegisterHandler
from handlers.add_contact_handler import AddContactHandler
from handlers.add_contact_by_login_handler import
AddContactByLoginHandler
from handlers.get_all_contacts_handler import GetAllContactsHandler
from handlers.confirm_contact_handler import ConfirmContactHandler
from handlers.chat_handler import ChatHandler
from handlers.create_multychat_handler import CreateMultyChatHandler

from database.database_service import DatabaseService

from services.register_service import RegisterService
from services.login_service import LoginService
from services.social_service import SocialService
from services.online_users_service import OnlineUsersService
from services.websocket_service import WebSocketService


def main():
   client = DatabaseService()
   online_users_service = OnlineUsersService()
   websocket_service = WebSocketService(online_users_service, client)
   register_service = RegisterService(client)
   login_service = LoginService(client, online_users_service)
   social_service = SocialService(online_users_service, client)


asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy(
))
   application = tornado.web.Application([
       (r"/login", LoginHandler, dict(login_service=login_service)),
                                    (r"/register",     RegisterHandler,
dict(register_service=register_service)),
                               (r"/add_contact",    AddContactHandler,
dict(social_service=social_service)),
                (r"/add_contact_by_login",  AddContactByLoginHandler,
dict(social_service=social_service)),
                      (r"/get_all_contacts",   GetAllContactsHandler,
dict(social_service=social_service)),
                       (r"/confirm_contact",   ConfirmContactHandler,
dict(social_service=social_service)),
                          (r"/create_chat",   CreateMultyChatHandler,
dict(social_service=social_service)),
          (r"/chat", ChatHandler, {'social_service': social_service,
'websocket_service': websocket_service})
   ])
```

```
    http_server = tornado.httpserver.HTTPServer(application)
    http_server.listen(8888)
    tornado.ioloop.IOLoop.instance().start()


if __name__ == '__main__':
    main()
```

Листинг Б.2 - Файл websocket_service.py

```
from messages.websocket_messages.init_message import InitMessage
from     messages.websocket_messages.send_message_to_server     import
SendMessageToServer
from services.online_users_service import OnlineUsersService
from database.database_service import DatabaseService
import tornado.websocket


class WebSocketService:
            def    init_handler(self,    message:    str,    socket:
tornado.websocket.WebSocketHandler):
        deserialized_message = InitMessage.deserialize(message)
         self._online_user_service.init_user(deserialized_message.jwt,
socket)
                                                unsended_messages      =
str(self._database_service.get_unsended_messages(

self._online_user_service.get_user_id(deserialized_message.jwt)))
        socket.write_message(
                        '{"type":  "init",  "unsended_messages":  %s}'  %
(unsended_messages)
        )
        # Удаление из БД

self._database_service.delete_unsended_messages(self._online_user_ser
vice.get_user_id(deserialized_message.jwt))

        def  message_to_server_handler(self,  message:  str,  socket:
tornado.websocket.WebSocketHandler):
                                             deserialized_message      =
SendMessageToServer.deserialize(message)
        receiver = deserialized_message.receiver
        # Receiver онлайн
                                                                      if
self._online_user_service.is_user_online(deserialized_message.receive
r):
            receiver_socket = self._online_user_service.get_socket(
                deserialized_message.receiver)
            receiver_socket.write_message(
                    '{"type": "to_client", "message": "%s", "sender":
"%s"}' %
                                          (deserialized_message.message,
self._online_user_service.get_user_id(deserialized_message.jwt))
            )
        # Receiver not online
```

```
        else:
            self._database_service.add_unsended_message(receiver,

self._online_user_service.get_user_id(deserialized_message.jwt),

deserialized_message.message)

         def  image_to_server_handler(self,  message:  str,  socket:
tornado.websocket.WebSocketHandler):
                                              deserialized_message      =
SendMessageToServer.deserialize(message)
        receiver = deserialized_message.receiver
        # Receiver онлайн
                                                                        if
self._online_user_service.is_user_online(deserialized_message.receive
r):
            receiver_socket = self._online_user_service.get_socket(
                deserialized_message.receiver)
            receiver_socket.write_message(
                        '{"type": "image_to_client", "message": "%s",
"sender": "%s"}' %
                                           (deserialized_message.message,
self._online_user_service.get_user_id(deserialized_message.jwt))
            )
        # Receiver not online
        else:
            self._database_service.add_unsended_message(receiver,

self._online_user_service.get_user_id(deserialized_message.jwt),

deserialized_message.message)

       def  __init__(self,  online_user_service:  OnlineUsersService,
database_service: DatabaseService):
        self._online_user_service = online_user_service
        self._database_service = database_service
        self.handlers = {
            'init': self.init_handler,
            'to_server': self.message_to_server_handler,
            'image': self.image_to_server_handler
        }
```

Листинг Б.3 - Файл social_service.py

```
from services.online_users_service import OnlineUsersService
from database.database_service import DatabaseService


class SocialService:
      def  __init__(self,  online_users_service:  OnlineUsersService,
database_service: DatabaseService):
        self._online_users_service = online_users_service
        self._database_service = database_service
```

```
    def get_user_id(self, jwt: str):
        return self._online_users_service.get_user_id(jwt)

    def add_contact(self, self_id, request_id):
            self._database_service.add_not_confirmed_contact(self_id,
request_id) # Добавляем в неподтвержденные
                    self._database_service.add_contact_request(self_id,
request_id) #Добавляем в запрос

    def add_contact_by_login(self, self_id, request_id):
        pass

    def get_all_contact(self, jwt: str):
        self_id = self._online_users_service.get_user_id(jwt)
        if self_id == -1:
            print("There is no id")
        contacts = self._database_service.get_contacts(self_id)
                                                requested_contacts      =
self._database_service.get_requested_contacts(self_id)
                                            not_confirmed_contacts      =
self._database_service.get_not_confirmed_contact(self_id)
        return contacts, requested_contacts, not_confirmed_contacts

    def confirm_contact(self, self_id, requested_id, answer):
         self._database_service.confirm_contact(self_id, requested_id,
answer)
```

Листинг Б.4 - Файл online_user_service.py

```
import tornado.websocket


class OnlineUserData:
    def __init__(self, id: str):
        self.id = id
        self.socket = None


class OnlineUsersService:
    def __init__(self):
        self.__users_online = dict()

    def add_user_online(self, jwt: str, id: str):
        self.__users_online[jwt] = OnlineUserData(id)
        #self.__users_online[jwt] = id

    def get_user_id(self, jwt: str):
                    return   self.__users_online[jwt].id   if   jwt   in
self.__users_online.keys() else -1

    def delete_user_online(self):
        pass
```

```
                def     init_user(self,     jwt:     str,     socket:
tornado.websocket.WebSocketHandler):
        self.__users_online[jwt].socket = socket

    def is_user_online(self, id):
        for v in self.__users_online.values():
            if v.id == id:
                return True
        return False

    def get_socket(self, id) -> tornado.websocket.WebSocketHandler:
        for v in self.__users_online:
            if self.__users_online[v].id == id:
                return self.__users_online[v].socket
        return None
```

Листинг Б.5 - Файл database_service.py

```
from pymongo import MongoClient
from bson.objectid import ObjectId
from common_types.enumerations import RegisterResults
from common_types.enumerations import LoginResults


class DatabaseService:
    def __init__(self):
        self.__mongo_client = MongoClient("localhost", 27017)
        self.__db = self.__mongo_client["chat_db"]
        self.__collection = self.__db["chat_db"]
        print(self.__db.name)

    def login(self, login: str, password: str):
        result = LoginResults.default
        id_object = -1
            user_entry  =  self.__collection.find_one({"login":  login,
"password": password})
        if user_entry is None:
            result = LoginResults.error
        else:
            result = LoginResults.success
            id_object = user_entry["_id"]
        return result, str(id_object)

    def register(self, login: str, password: str):
        result = RegisterResults.default
        id_object = -1
        user_entry = self.__collection.find_one({"login": login})
        if user_entry is None:
            id_object = self.__collection.insert_one(
                {"login": login, "password": password,
                             "contacts":  [],  "contacts_requests":  [],
"not_confirmed_contact": [],
                 "unsended_messages": {}}).inserted_id
            result = RegisterResults.success
```

```
        else:
            result = RegisterResults.login_busy
        return result, str(id_object)

    # У пользователя, который послал заявку
    def add_not_confirmed_contact(self, self_id, request_id):
          # todo возможно косяк, из-за несоответствия типов id. Если
так, то руками в бд добавлять id
        self.__collection.update({"_id": ObjectId(self_id)}, {"$push":
{"not_confirmed_contact": request_id}})

    # У пользователя, которому послали заявку
    def add_contact_request(self, self_id, request_id):
              self.__collection.update({"_id":  ObjectId(request_id)},
{"$push": {"contacts_requests": self_id}})

    def get_contacts(self, self_id):
                           return   self.__collection.find_one(({"_id":
ObjectId(self_id)}))["contacts"]

    def get_requested_contacts(self, self_id):
                           return   self.__collection.find_one(({"_id":
ObjectId(self_id)}))["contacts_requests"]

    def get_not_confirmed_contact(self, self_id):
                           return   self.__collection.find_one(({"_id":
ObjectId(self_id)}))["not_confirmed_contact"]

    # Вызывается для пользователя, которому послали заявку
    def confirm_contact(self, self_id, requested_id, answer):
        # Удалили из полученных заявок
        self.__collection.update({"_id": ObjectId(self_id)}, {"$pull":
{"contacts_requests": str(requested_id)}})
        # Удалили из отправленных заявок
            self.__collection.update({"_id":  ObjectId(requested_id)},
{"$pull": {"not_confirmed_contact": str(self_id)}})
        if answer == 1:
            # добавить друг друга в контакты
                 self.__collection.update({"_id":  ObjectId(self_id)},
{"$push": {"contacts": requested_id}})
             self.__collection.update({"_id": ObjectId(requested_id)},
{"$push": {"contacts": self_id}})

    def get_unsended_messages(self, self_id):
                           return   self.__collection.find_one(({"_id":
ObjectId(self_id)}))["unsended_messages"]

    def add_unsended_message(self, receiver_id, sender_id, message):
              self.__collection.update({"_id":  ObjectId(receiver_id)},
{"$push": {"unsended_messages.{}".format(sender_id): message}})

    def delete_unsended_messages(self, self_id):
         self.__collection.update({"_id": ObjectId(self_id)}, {"$set":
{"unsended_messages": {}}})
```

Листинг Б.6 - Файл jwt_service.py

```
import jwt
from datetime import datetime, timedelta


JWT_SECRET = "jwt_secret_code"
JWT_ALGORITHM = "HS256"
JWT_EXP_DELTA_SECONDS = 86400

def create_jwt(id):
   payload = {
       "user_id": id,
                                       "exp":     datetime.utcnow()     +
timedelta(seconds=JWT_EXP_DELTA_SECONDS)
   }
   jwt_token = jwt.encode(payload, JWT_SECRET, JWT_ALGORITHM)
   return str(jwt_token.decode('utf-8'))
```

Листинг Б.7 - Файл encryption_service.py

```
import hashlib


class EncryptionService:
   @staticmethod
   def get_hash(input_string):
       return hashlib.md5(input_string).hexdigest()
```

Листинг Б.8 - Файл login_service.py

```
from database.database_service import DatabaseService
from messages.login_message import LoginMessage
from common_types.enumerations import LoginResults
from services import jwt_service
from services.online_users_service import OnlineUsersService
import json


class LoginService:
         def   __init__(self,   database_service:   DatabaseService,
online_users_service: OnlineUsersService):
       self._database_service = database_service
       self._online_users_service = online_users_service

   def login(self, message: LoginMessage):
         db_result, id = self._database_service.login(message.login,
message.password)
       if db_result == LoginResults.success:
           jwt = jwt_service.create_jwt(id)
           response = dict({"result": "success", "jwt": str(jwt)})
           self._online_users_service.add_user_online(jwt, id)
```

```
            elif db_result == LoginResults.error or db_result ==
LoginResults.not_validated or db_result == LoginResults.server_error:
            response = dict({"result": "failed"})
        return response
```

Листинг Б.9 - Файл register_service.py

```
from database.database_service import DatabaseService
from messages.register_message import RegisterMessage
from common_types.enumerations import RegisterResults
from services import jwt_service
import json


class RegisterService:
    def __init__(self, database_service: DatabaseService):
        self._database_service = database_service

    def register(self, message: RegisterMessage):
        db_result, id = self._database_service.register(message.login,
message.password)
        if db_result == RegisterResults.success:
            jwt = jwt_service.create_jwt(id)
            response = dict({"result": "success", "jwt": str(jwt)})
         elif db_result == RegisterResults.login_busy or db_result ==
RegisterResults.not_validated or db_result ==
RegisterResults.server_error:
            print(db_result)
            response = dict({"result": "failed"})
        return response
```

Листинг Б.10 - Файл register_handler.py

```
import tornado.web
from messages.register_message import RegisterMessage
from services.register_service import RegisterService
import json


class RegisterHandler(tornado.web.RequestHandler):
    def prepare(self):
        header = "Content-Type"
        body = "application/json"
        self.set_header(header, body)

    def initialize(self, register_service) -> None:
        self._register_service = register_service

    def post(self):
        try:
                                                        message     =
RegisterMessage.deserialize(self.request.body.decode('utf-8'))
        except:
```

```
            self.set_status(400)
            self.write(dict({"result": "failed"}))
            self.finish()
            return
        result = self._register_service.register(message)
        if result["result"] == "success":
            self.set_status(200)
        else:
            self.set_status(409)
        self.set_header("Content-Type", "application/json")
        self.write(result)
        self.finish()

    def get(self):
        pass
```

Приложение В

ИСХОДНЫЙ КОД КЛИЕНТСКОЙ ЧАСТИ

(рекомендуемое)

Листинг В.1 - Файл AppManager.cs

```
using System.Collections;
using System;
using System.Collections.Generic;
using UnityEngine;
using Chat.Managers;
using Chat.UI;
using Chat.Network;

namespace Chat
{
    public class AppManager : MonoBehaviour
    {
        [SerializeField]
        private PageController pageController;


        private NetworkManager networkManager;
        private Session session;
        private FileManager fileManager;
        private MessagesManager messagesManager;
        private JsonDecoder jsonDecoder;
        private PageManager pageManager;

        private void Start()
        {
            jsonDecoder = new JsonDecoder();
            session = new Session();
                   networkManager = new NetworkManager(jsonDecoder,
session);
            fileManager = new FileManager();
                  messagesManager = new MessagesManager(fileManager,
jsonDecoder);

            PageManager.Configure(
                pageController.LoginPage,
                pageController.RegisterPage,
                pageController.MainPage,
                pageController.DialogPage,
                pageController.AddContactPage,
                pageController.RequestedContactPage);


pageController.LoginPage.GetComponent<LoginPageController>().Configur
e(networkManager);

pageController.RegisterPage.GetComponent<RegisterPageController>().Co
nfigure(networkManager);

pageController.MainPage.GetComponent<MainPageController>().Configure(
networkManager, messagesManager);

pageController.DialogPage.GetComponent<DialogPageController>().Config
ure(networkManager);
```

```
pageController.AddContactPage.GetComponent<AddContactPageController>(
).Configure(networkManager);

pageController.RequestedContactPage.GetComponent<RequestedContactsPag
eController>().Configure(networkManager);
        }
    }
}
```

Листинг В.2 - Файл NetworkManager.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using Chat.Network;
using Chat.Messages;
using Chat.Messages.Responses;
using Chat.Configs;
using Chat.Messages.Websocket;
using System.Threading.Tasks;

namespace Chat.Managers
{
    public class NetworkManager
    {
        private JsonDecoder jsonDecoder;
        private HttpNetworkService httpNetworkService;
        private WebSocketAdapter webSocketAdapter;
        private Dispatcher dispatcher;
        private WebSocketService webSocketService;

        private Session session;

        private string baseLobbyUrl;
        //private string websocketUrl;

             public  NetworkManager(JsonDecoder  jsonDecoder,  Session
session)
        {
            this.jsonDecoder = jsonDecoder;
            httpNetworkService = new HttpNetworkService();
            baseLobbyUrl = string.Concat(NetworkConfig.LobbyProtocol,
                NetworkConfig.MasterAdress,
                NetworkConfig.ServerPort);
            webSocketAdapter = new WebSocketAdapter(string.Concat(
                NetworkConfig.WebsocketProtocol,
                NetworkConfig.MasterAdress,
                NetworkConfig.ServerPort,
                NetworkConfig.WebsocketUrl));
            dispatcher = new Dispatcher();
            webSocketService = new WebSocketService(webSocketAdapter,
jsonDecoder);
```

```
            webSocketService.SubscribeToMethod<UserMessage>((sender,
messageArg) => dispatcher.Invoke(() =>
                GetUserMessage(messageArg.Message)));
            this.session = session;
        }

                public  async  Task<AuthResponse>  Login(AuthMessage
authorizeMessage)
        {
            var result = await httpNetworkService.Post(
                jsonDecoder.Serialize(authorizeMessage),
                string.Concat(baseLobbyUrl, NetworkConfig.LoginUrl)
            );
            var response = new AuthResponse();
            if (result.Status == 200)
            {
                                                        response   =
jsonDecoder.Deserialize<AuthResponse>(result.Message,
MessagesTypes.LoginResponse);
                response.IsSuccesful = true;
            }
            return response;
        }

                            public   async   Task<AddContactResponse>
AddContact(AddContactMessage addContactMessage)
        {
            var result = await httpNetworkService.Post(
                jsonDecoder.Serialize(addContactMessage),
                                        string.Concat(baseLobbyUrl,
NetworkConfig.AddContactUrl),
                session.Jwt);
            var response = new AddContactResponse();
            if (result.Status == 200)
            {
                                                      response   =
jsonDecoder.Deserialize<AddContactResponse>(result.Message,
MessagesTypes.AddContactResponse);
                response.IsSuccesful = true;
            }
            return response;
        }

        public async Task<GetAllContactsResponse> GetContacts()
        {
            var result = await httpNetworkService.Get(
                                        string.Concat(baseLobbyUrl,
NetworkConfig.GetAllContactsUrl),
                session.Jwt);
            var response = new GetAllContactsResponse();
            if (result.Status == 200)
            {
                                                      response   =
jsonDecoder.Deserialize<GetAllContactsResponse>(result.Message,
MessagesTypes.GetContactsResponse);
```

```
                response.IsSuccesful = true;
            }
            return response;
        }

         public async Task<RegisterResponse> Register(RegisterMessage
authorizeMessage)
        {
            var result = await httpNetworkService.Post(
                jsonDecoder.Serialize(authorizeMessage),
                                        string.Concat(baseLobbyUrl,
NetworkConfig.RegisterUrl)
            );
            var response = new RegisterResponse();
            if (result.Status == 200)
            {
                                                          response    =
jsonDecoder.Deserialize<RegisterResponse>(result.Message,
MessagesTypes.RegisterResponse);
                response.IsSuccesful = true;
            }
            return response;
        }

                        public   async   Task<ConfirmContactResponse>
ConfirmContact(ConfirmContactMessage message)
        {
            var result = await httpNetworkService.Post(
                jsonDecoder.Serialize(message),
                                        string.Concat(baseLobbyUrl,
NetworkConfig.ConfirmContactUrl),
                session.Jwt);
            var response = new ConfirmContactResponse();
            if (result.Status == 200)
            {
                                                        response    =
jsonDecoder.Deserialize<ConfirmContactResponse>(result.Message,
MessagesTypes.ConfirmContactResponse);
                response.IsSuccesful = true;
            }
            return response;
        }

                        public   async   Task<CreateMultyChatResponse>
CreateMultyChat(CreateMultyChatMessage message)
        {
            var result = await httpNetworkService.Post(
                jsonDecoder.Serialize(message),
                                        string.Concat(baseLobbyUrl,
NetworkConfig.AddContactUrl),
                session.Jwt);
            var response = new CreateMultyChatResponse();
            if (result.Status == 200)
            {
```

```
                                                                response    =
jsonDecoder.Deserialize<CreateMultyChatResponse>(result.Message,
MessagesTypes.GetContactsRequest);
                response.IsSuccesful = true;
            }
            return response;
        }

        public async void OpenConnection()
        {

        }

        private void GetUserMessage(UserMessage message)
        {

        }
    }
}
```

Листинг В.3 - Файл EncryptionService.cs

```
using System;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Text;
using Chat.Configs;

namespace Chat.Managers
{
    public static class EncryptionService
    {
        public static string Encrypt(string text)
        {
                        using  (MD5CryptoServiceProvider  md5  =  new
MD5CryptoServiceProvider())
            using (var tdes = new TripleDESCryptoServiceProvider())
            {
                                                                tdes.Key    =
md5.ComputeHash(UTF8Encoding.UTF8.GetBytes(SecureConfig.MD5Key));
                tdes.Mode = CipherMode.ECB;
                tdes.Padding = PaddingMode.PKCS7;

                using (var transform = tdes.CreateEncryptor())
                {
                                                    byte[]   textBytes   =
UTF8Encoding.UTF8.GetBytes(text);
                                                       byte[]   bytes   =
transform.TransformFinalBlock(textBytes, 0, textBytes.Length);
                                return  Convert.ToBase64String(bytes,  0,
bytes.Length);
                }
            }
        }
```

```
        public static string Decrypt(string cipher)
        {
            using (var md5 = new MD5CryptoServiceProvider())
            using (var tdes = new TripleDESCryptoServiceProvider())
            {

                                                                tdes.Key    =
md5.ComputeHash(UTF8Encoding.UTF8.GetBytes(SecureConfig.MD5Key));
                tdes.Mode = CipherMode.ECB;
                tdes.Padding = PaddingMode.PKCS7;

                using (var transform = tdes.CreateDecryptor())
                {
                                                    byte[]   cipherBytes   =
Convert.FromBase64String(cipher);
                                                          byte[]   bytes   =
transform.TransformFinalBlock(cipherBytes, 0, cipherBytes.Length);
                    return UTF8Encoding.UTF8.GetString(bytes);
                }
            }
        }
    }
}
```

Листинг В.4 - Файл FileManager.cs

```
using UnityEngine;

namespace Chat.Managers
{
    public class FileManager
    {
        public bool Exists(string key)
        {
            return PlayerPrefs.HasKey(key);
        }

        public void Delete(string key)
        {
            PlayerPrefs.DeleteKey(key);
        }

        public string LoadText(string key)
        {
            return PlayerPrefs.GetString(key);
        }

        public void SaveText(string key, string text)
        {
            PlayerPrefs.SetString(key, text);
        }
    }
}
```

Листинг В.5 - Файл MessagesManager.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using Chat.Models;
using System.Threading.Tasks;

namespace Chat.Managers
{
    public class MessagesManager
    {
        private FileManager fileManager;
        private Network.JsonDecoder jsonDecoder;
        private string messageFileName;

        public MessagesModel MessagesModel { get; set; }

        public MessagesManager(FileManager fileManager,
Network.JsonDecoder jsonDecoder)
        {
            this.jsonDecoder = jsonDecoder;
            this.fileManager = fileManager;
            messageFileName = Configs.SecureConfig.PrefsKey;
        }

        public MessagesModel GetMessageHistory()
        {
            MessagesModel messages;
            //не первый раз заходит или не удалял историю
            if (fileManager.Exists(messageFileName))
            {
                string hashMessageText =
fileManager.LoadText(messageFileName);
                //string messageText =
EncryptionService.Decrypt(hashMessageText);
```

```
                messages =
jsonDecoder.Deserialize<MessagesModel>(hashMessageText,
Messages.MessagesTypes.MessagesModel);
            }
            //первый раз или удалил историю
            else
            {
                messages = new MessagesModel()
                {
                    Messages = new List<List<string>>()
                };
            }
            MessagesModel = messages;
            return messages;
        }

        public void SaveMessageHistory()
        {
            string messageJson =
jsonDecoder.Serialize(MessagesModel);
            //todo зашифровать
            if (fileManager.Exists(messageFileName))
                fileManager.Delete(messageFileName);
            fileManager.SaveText(messageFileName, messageJson);
        }
    }
}
```

Листинг В.6 - Файл HttpNetworkService.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using System.Net.Http;
using System.IO;
```

```
namespace Chat.Network
{
    public class HttpNetworkService
    {
        //todo получить конфиги

        public async Task<HttpRequestResult> Post(string jsonMessage,
string uri, string authToken = null)
        {
            UnityEngine.Debug.Log($"Sended http request -
{jsonMessage}");
            var data = new StringContent(jsonMessage, Encoding.UTF8,
"application/json");
            using (var client = new HttpClient())
            {
                if (!string.IsNullOrEmpty(authToken))
                    client.DefaultRequestHeaders.Add("JWT",
authToken);

                var response = await client.PostAsync(uri, data);
                string result =
response.Content.ReadAsStringAsync().Result;
                var code = response.StatusCode;
                UnityEngine.Debug.Log($"Received http response -
{result}");
                return new HttpRequestResult((int)code, result);
            }
        }

        //todo исправить
        public async Task<HttpRequestResult> Get(string uri, string
authToken)
        {
            using (var client = new HttpClient())
            {
                if (!string.IsNullOrEmpty(authToken))
                    client.DefaultRequestHeaders.Add("JWT",
authToken);
```

```
                var response = await client.GetAsync(uri);
                string result =
response.Content.ReadAsStringAsync().Result;
                var code = response.StatusCode;
                return new HttpRequestResult((int)code, result);
            }
        }

    }
}
```

Листинг В.7 - Файл JsonDecoder.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using Newtonsoft.Json;
using Chat.Messages;
using Chat.Messages.Responses;

namespace Chat.Network
{
    public class JsonDecoder
    {
        private readonly Dictionary<string, Func<string, IMessage>>
deserializedMessage = new Dictionary<string, Func<string,
IMessage>>()
        {
            [MessagesTypes.LoginResponse] = new Func<string,
IMessage>(LoginResponse),
            [MessagesTypes.MessagesModel] = new Func<string,
IMessage>(MessagesModel),
            [MessagesTypes.RegisterResponse] = new Func<string,
IMessage>(RegisterResponse),
            [MessagesTypes.AddContactResponse] = new Func<string,
IMessage>(AddContactResponse),
        };
```

```
        public string Serialize<T>(T request) where T : IMessage
        {
            return JsonConvert.SerializeObject(request);
        }

        public T Deserialize<T>(string response, string type = "")
where T : IMessage
        {
            if (type == "")
            {
                Message message =
JsonConvert.DeserializeObject<Message>(response);
                var deserializedObject =
deserializedMessage[message.Type].Invoke(response);
                return (T)deserializedObject;
            }
            return (T)deserializedMessage[type].Invoke(response);
        }

        public string GetType(string message)
        {
            return
JsonConvert.DeserializeObject<Message>(message).Type;
        }

        private static AuthResponse LoginResponse(string response)
        {
            return
JsonConvert.DeserializeObject<AuthResponse>(response);
        }

        private static Models.MessagesModel MessagesModel(string
response)
        {
            return
JsonConvert.DeserializeObject<Models.MessagesModel>(response);
        }
```

```
        private static RegisterResponse RegisterResponse(string
response)
        {
            return
JsonConvert.DeserializeObject<RegisterResponse>(response);
        }

        private static AddContactResponse AddContactResponse(string
response)
        {
            return
JsonConvert.DeserializeObject<AddContactResponse>(response);
        }
    }
}
```

Листинг В.8 - Файл WebSocketAdapter.cs

```
using System;
using WebSocketSharp;

namespace Chat.Network
{
    public class WebSocketAdapter
    {
        public bool IsConnected { get; private set; }

        private WebSocket socket;

        public WebSocketAdapter(string uri)
        {
            socket = new WebSocket(uri);
            socket.OnClose += (sender, args) => OnClose?.Invoke(this,
args);
            socket.OnOpen += (sender, args) => OnOpen?.Invoke(this,
args);
```

```
            socket.OnMessage += (sender, args) =>
OnMessage?.Invoke(this, args);
            socket.OnError += (sender, args) => OnError?.Invoke(this,
args);
            socket.Log.Level = LogLevel.Trace;
            socket.Log.File = "C:\\build\\log.txt";
        }

        public event EventHandler<ErrorEventArgs> OnError;
        public event EventHandler<CloseEventArgs> OnClose;
        public event EventHandler<MessageEventArgs> OnMessage;
        public event EventHandler OnOpen;

        public void Close()
        {
            socket.Close();
        }

        public void Connect()
        {
            try
            {
                socket.Connect();
                IsConnected = true;
            }
            catch
            {
                IsConnected = false;
            }
        }

        public void Send(string data)
        {
            socket.Send(data);
        }
    }
}
```

Листинг В.9 - Файл WebSocketService.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using WebSocketSharp;
using Chat.Configs;
using Chat.CommonTypes;
using Chat.Messages;
using Chat.Messages.Websocket;
using UnityEngine;

namespace Chat.Network
{
    public class WebSocketService
    {
        private Dictionary<Type, IPublisher> publishers;
        private Dictionary<string, Action<string>> messageHandlers;
        private WebSocketAdapter webSocket;
        private JsonDecoder jsonDecoder;

        public event EventHandler OnConnected;
        public event EventHandler OnMessage;
        public event EventHandler OnError;
        public event EventHandler OnClose;

        public bool IsConnected { get; private set; }

        public WebSocketService(WebSocketAdapter webSocketAdapter,
JsonDecoder jsonDecoder)
        {
            webSocket = webSocketAdapter;
            this.jsonDecoder = jsonDecoder;

            webSocket.OnMessage += OnMessageReceive;
            webSocket.OnOpen += (sender, e) => { IsConnected = true;
OnConnected?.Invoke(this, e); };
            webSocket.OnError += OnErrorReceive;
```

```
            webSocket.OnClose += (sender, e) => { IsConnected =
false; OnClose?.Invoke(this, e); };

            publishers = new Dictionary<Type, IPublisher>()
            {
                [typeof(UserMessage)] = new Publisher<UserMessage>()
            };
            messageHandlers = new Dictionary<string,
Action<string>>()
            {
                [MessagesTypes.UserMessageReceive] =
HandleMessage<UserMessage>
            };
        }

        public void SendMessage(string message)
        {
            if (IsConnected && webSocket != null)
            {
                webSocket.Send(message);
                Debug.Log($"Sended websocket message {message}");
            }
            else
            {
                throw new
Exception("NetworkManager.Instance.NetworkController.WebSocket is
closed");
            }
        }

        public void Connect()
        {
            webSocket.Connect();
        }

        public void OnConnectionClose(object sender, CloseEventArgs
e)
        {
```

```
            //todo
        }

        protected void OnMessageReceive(object sender,
MessageEventArgs e)
        {
            UnityEngine.Debug.Log($"Receive message {e.Data}");
            try
            {
                string type = jsonDecoder.GetType(e.Data);
                messageHandlers[type](e.Data);
            }
            catch (Exception ex)
            {
                Debug.LogError($"Got error while handling message.
\n{ex.Message}\n{ex.StackTrace}");
                throw;
            }
        }

        protected void OnErrorReceive(object sender, ErrorEventArgs
e)
        {
            if (e.Message.Contains("WebSocket connection has already
been closed"))
            {
                Debug.LogWarning("Error: " + e.Message);
            }
            else
            {
                throw new ApplicationException("Error: " +
e.Message);
            }
        }

        private void HandleMessage<T>(string message) where T :
IMessage
        {
```

```
            PublishData(jsonDecoder.Deserialize<T>(message));
        }

        private void PublishData<T>(T message) where T : IMessage
        {

((IPublisher<T>)publishers[typeof(T)]).PublishData(message);
        }

        public void
SubscribeToMethod<T>(EventHandler<MessageReceivedEventArgs<T>>
handler)
        {
            ((IPublisher<T>)publishers[typeof(T)]).DataPublisher +=
handler;
        }
    }
}
```

Листинг В.10 - Файл PageManager.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using Chat.CommonTypes;
using Chat.Managers;
using Chat.UI;
using UnityEngine;

namespace Chat.Managers
{
    public class PageManager
    {

        private static Dictionary<PageTypes, GameObject> pages;
        private static PageTypes currentPage = PageTypes.Auth;
```

```
        public static void Configure(GameObject loginPage, GameObject
registerPage, GameObject mainPage, GameObject dialogPage,
            GameObject addContactPage, GameObject contactsPage)
        {
            pages = new Dictionary<PageTypes, GameObject>()
            {
                [PageTypes.Auth] = loginPage,
                [PageTypes.Register] = registerPage,
                [PageTypes.Main] = mainPage,
                [PageTypes.Dialog] = dialogPage,
                [PageTypes.AddContact] = addContactPage,
                [PageTypes.Contacts] = contactsPage
            };
        }

        public static void TransitionScreen(PageTypes requestPage,
bool isModal=false)
        {
            pages[currentPage].SetActive(false);
            currentPage = requestPage;
            pages[currentPage].SetActive(true);

pages[currentPage].GetComponent<IPageController>().OnOpenPage();
        }
    }
}
```

Листинг В.11 - Файл Dispatcher.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Chat.Network
{
    public class Dispatcher
```

```
    {
        public List<Action> pendingActions = new List<Action>();

        public void Invoke(Action action)
        {
            lock (pendingActions)
            {
                pendingActions.Add(action);
            }
        }

        public void InvokePending()
        {
            lock (pendingActions)
            {
                foreach (var action in pendingActions)
                {
                    action();
                }

                pendingActions.Clear();
            }
        }
    }
}
```

Листинг В.12 - Файл Publisher.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Chat.Network
{
    public interface IPublisher
    {
    }
```

```
      public interface IPublisher<T> : IPublisher
      {
            event EventHandler<MessageReceivedEventArgs<T>>
DataPublisher;
            void OnDataPublisher(MessageReceivedEventArgs<T> args);
            void PublishData(T data);
      }

    public class Publisher<T> : IPublisher<T>
    {
        public event EventHandler<MessageReceivedEventArgs<T>>
DataPublisher;

        public void OnDataPublisher(MessageReceivedEventArgs<T> args)
        {
            DataPublisher?.Invoke(this, args);
        }

        public void PublishData(T data)
        {
            MessageReceivedEventArgs<T> message = new
MessageReceivedEventArgs<T>(data);
            OnDataPublisher(message);
        }
    }

    public class MessageReceivedEventArgs<T> : EventArgs
      {
            public T Message { get; }

            public MessageReceivedEventArgs(T message)
            {
                  Message = message;
            }
      }
}
```

Приложение Г

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ СЕРВЕРНОЙ ЧАСТИ

(рекомендуемое)

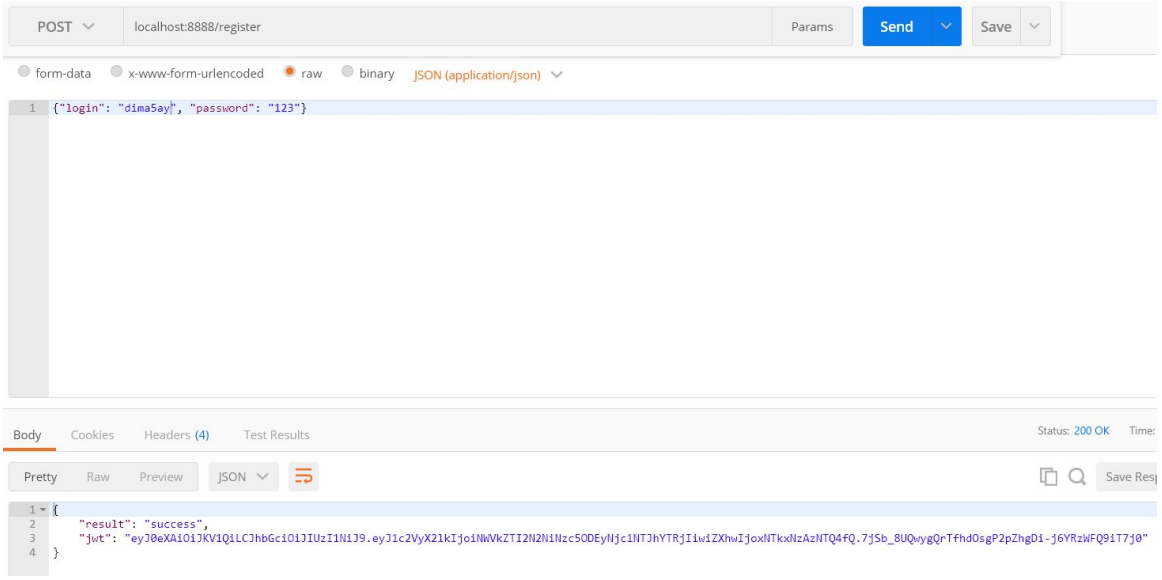

Рисунок Г.1 - Успешная регистрация

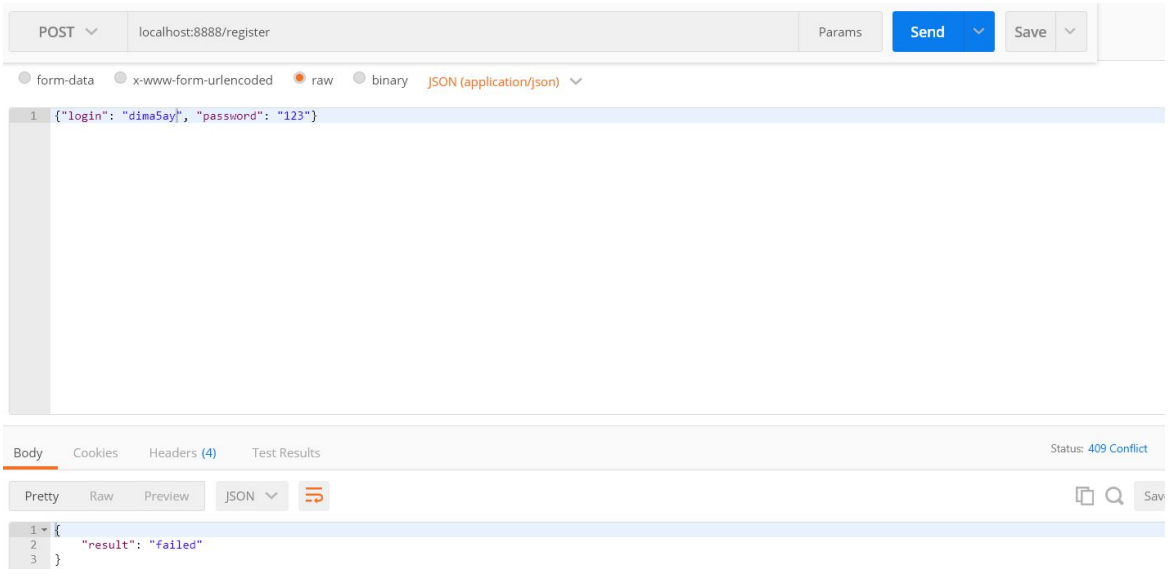

Рисунок Г.2 - Попытка зарегистрироваться логином, уже находящимся в базе данных

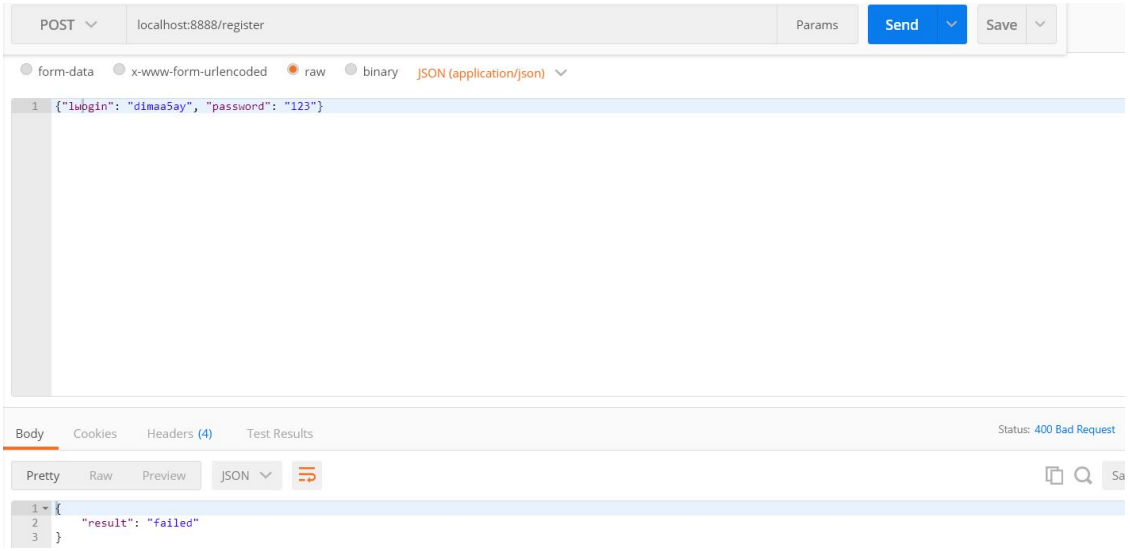

Рисунок Г.3 - Попытка зарегистрироваться, отправив несуществующий параметр

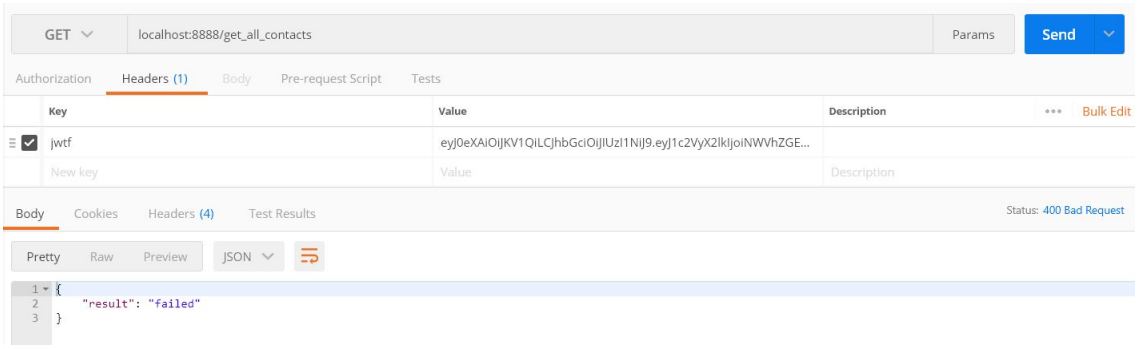

Рисунок Г.4 - Попытка получить все контакты, отправив несуществующий параметр вместо JWT

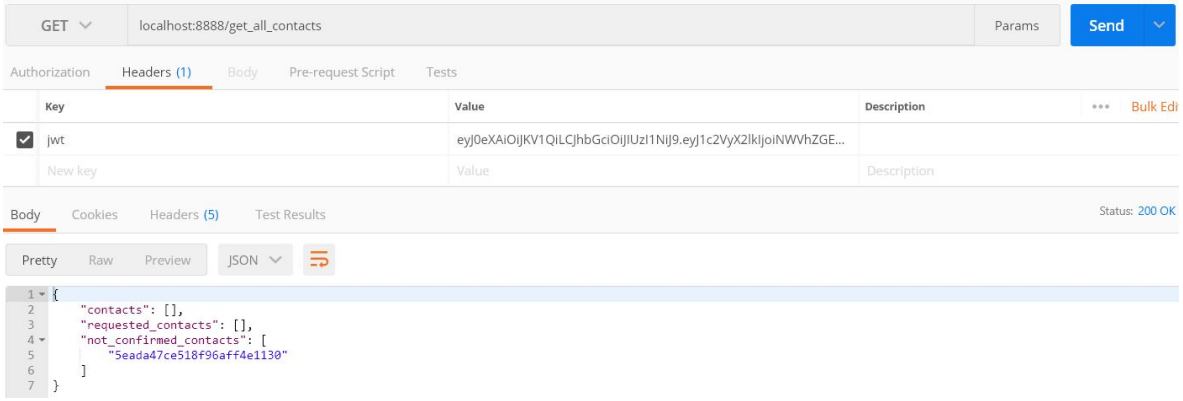

Рисунок Г.5 - Успешная попытка получить все контакты

{"type": "init", "jwt": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiNWVhZGE0N2NlNTE4Zjk2YWZmNGUxMTMwIiwiZXhwIjoxNTkyMjM4MDE1fQ.Leu3bEDR4XhccEGTkwxIVK01hgTeKUUiAPsEBob38Fk"}
{"type": "init", "unsended_messages": {}}
{"type": "to_server", "jwt": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiNWVhZGE0N2NlNTE4Zjk2YWZmNGUxMTMwIiwiZXhwIjoxNTkyMjM4MDE1fQ.Leu3bEDR4XhccEGTkwxIVK01hgTeKUUiAPsEBob38Fk",
"receiver": "5eada482e518f96aff4e1131",
"message": "hello3"}

Рисунок Г.6 - Попытка первого клиента отправить сообщение

{
"type": "to_client",
"message": "hello3",
"sender": "5eada47ce518f96aff4e1130"
}

Рисунок Г.7 - Второй клиент успешно получает сообщение

Приложение Д

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

(рекомендуемое)

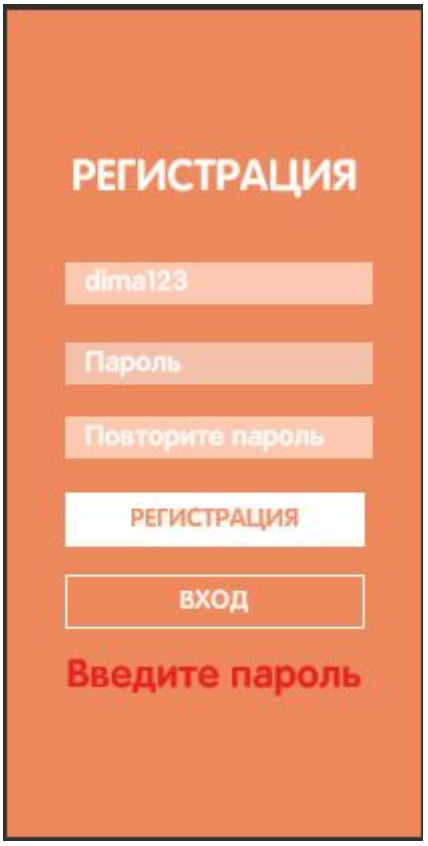

Рисунок Д.1 - Тест ввода пустых значений

Рисунок Д.2 - Тест использования символов кириллицы в пароле

Рисунок Д.3 - Тест ввода некорректных символов в пароле

Рисунок Д.4 - Тест ввода пароля длиной менее 7 символов

Рисунок Д.5 - Тест ввода корректных данных

Рисунок Д.6 - Тест ввода несовпадающих паролей