

## РЕФЕРАТ

Пояснительная записка – 31 с, 11 рис., 0 табл., 4 источника.

ВЕБ СЕРВИС, ASP.NET, REACT, TYPESCRIPT, SIGNALR, GRPC,  
МЕССЕНДЖЕР, RABBITMQ, ENTITY FRAMEWORK

В наше время современный бизнес требует активного общения с клиентами в удобной для них среде. На данный момент самым удобным способом для общения люди выбирают мессенджеры.

Данное приложение позволяет людям общаться из одного окна через несколько мессенджеров.

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	6
1 Постановка и анализ задачи. ....	7
1.1 Описание предметной области.....	7
1.2 Диаграмма Вариантов использования.....	8
1.3 Обоснование выбора средств реализации.....	9
2 Анализ данных.....	12
2.1 Логическая структура данных.....	12
3 Программная реализация.....	14
3.1 Архитектура приложения .....	14
3.2 Реализация клиентской части.....	15
3.3 Реализация серверной части.....	15
3.4 Анализ алгоритмов.....	17
4 Тестирование.....	20
5 Техническое задание .....	22
6 Руководство пользователя.....	24
ЗАКЛЮЧЕНИЕ .....	26
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ .....	27
ПРИЛОЖЕНИЯ .....	28

## ВВЕДЕНИЕ

На сегодняшний день все больше и больше предприятий переходят на мессенджеры для общения с клиентами, отказываясь от звонков. И действительно, мессенджеры – это наиболее удобное средство связи, так как позволяет отвечать на сообщения в удобное время и отвечать таким способом, как пользователю удобно.

Но мессенджеров довольно много, что приводит к многим неудобствам для отдела общения с клиентами:

- множество приложений-мессенджеров, среди которых можно запутаться;
- невозможность открыть один аккаунт на нескольких устройствах одновременно.

Разработанное клиент-серверное приложение позволяет избежать данных проблем.

# **1 Постановка и анализ задачи.**

## **1.1 Описание предметной области**

В наше время современный бизнес требует активного общения с клиентами в удобной для них среде. На данный момент самым удобным способом для общения люди выбирают мессенджеры. И действительно: проще написать кому-либо или оставить голосовое сообщение, а затем ответить, когда появится время и придет уведомление.

Клиенты – простые люди со своими вкусами, предпочтениями и кругом общения. И если сейчас взглянуть на список популярных мессенджеров, то мы обнаружим что их не один или два, а десятки. А отвечать только в одном – значит потерять клиентскую базу, которым просто будет неудобно написать в выбранном мессенджере. Но даже если выделить несколько наиболее популярных мессенджеров в сфере продаж и работать только с ними, то остаются следующие проблемы:

- нет возможности у некоторых популярных мессенджеров открывать несколько приложений на разных компьютерах для одного номера телефона;
- нет группировки чатов по темам;
- хранение переписки и важной информации зависит от компании, предоставляющей мессенджер.

Для решений этих проблем можно соединить всё необходимое в одном месте, в приложении - агрегаторе. На данный момент существует множество аналогов, например, Umnico, МультиЧат, ЕАДЕСК, Chat2Desk. Данные агрегаторы предоставляют возможность общаться с клиентами через различные мессенджеры и отслеживать статистику менеджеров. Для общения в них необходимо покупать аккаунт для каждого отдельного менеджера. Также для доступа к разным мессенджерам необходимо платить ежемесячную плату.

## 1.2 Диаграмма Вариантов использования

Управление агрегатором мессенджеров производится не только непосредственно операторами, которые отвечают на сообщения клиентов. Также необходимы администраторы для контроля качества операторов, регистрации аккаунтов и других руководящих целей. Для этого разработаны два уровня доступа к данным. На диаграмме использования их можно разделить на два действующих лица:

- менеджер;
- администратор.

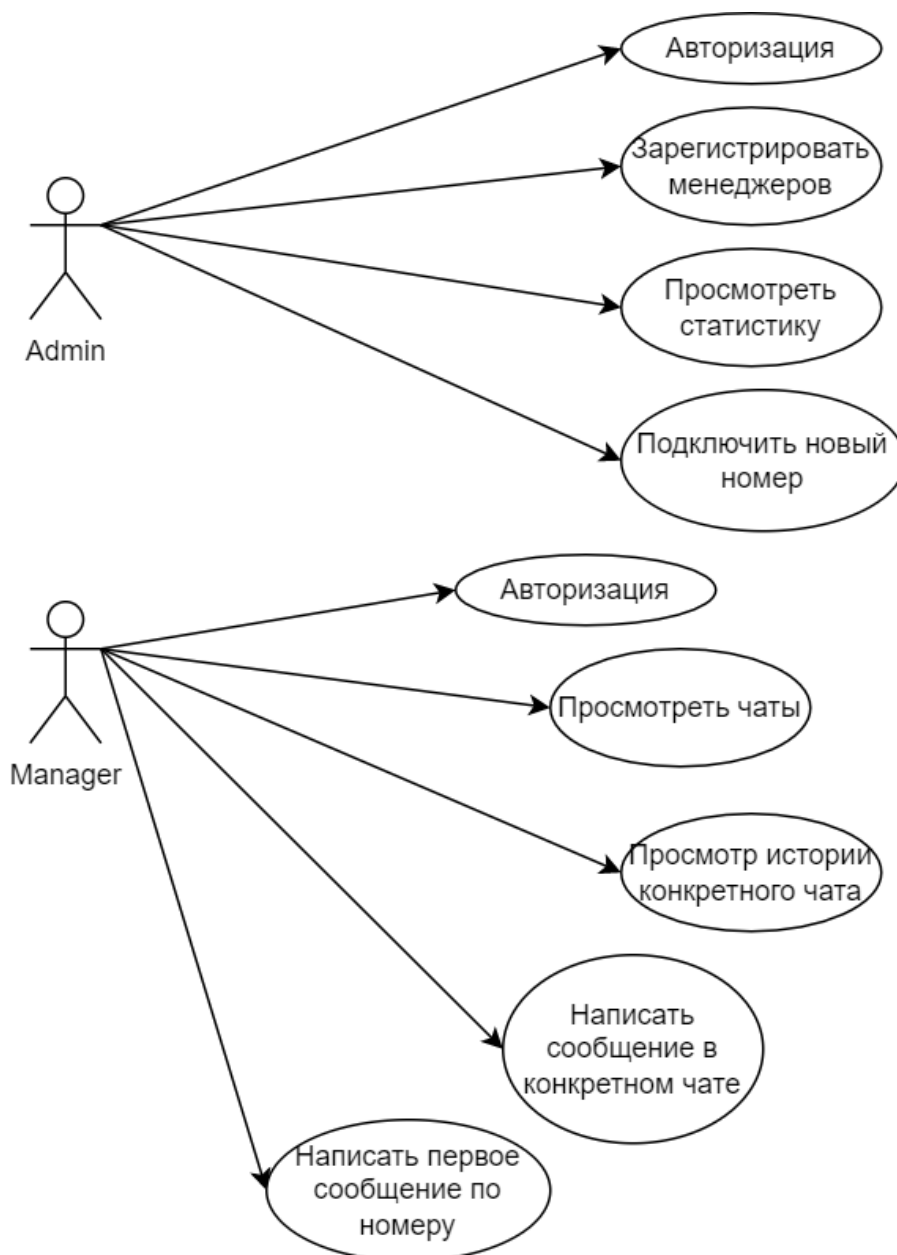


Рисунок 1- Диаграмма вариантов использования

Для работы программы требуется реализовать:

- регистрация номеров/аккаунтов мессенджеров;
- отправка/получение сообщений;
- регистрация менеджеров в системе;
- отслеживание статистики.

### **1.3 Обоснование выбора средств реализации**

Для того, чтобы понять, как разрабатывать данный проект, первым делом потребуется изучить API актуальных мессенджеров. Тут ставится задача по определению возможностей API. Например, популярный мессенджер Viber предоставляет Viber REST API, который по своей сути является ботом. А значит имеет ограничение на отправление сообщений первым, то есть клиент должен написать первым, чтобы диалог состоялся. Это не всегда удобно, но другого выбора в Viber нет. Также бот имеет ограничение на отправление бесплатных сообщений в 25000, что может послужить причиной увеличения стоимости пользования этим мессенджером.

Наиболее большой выбор для разработчика предоставляет Telegram: на сайте представлено три вида API для использования. Первый и самый простой – это схожий с Viber ботом Bot API. Имеет тоже очень важное ограничение на отправление сообщений первым. Второй способ и самый сложный – это Telegram API. Этот API позволяет создать собственный клиент Telegram. Он на 100% открыт для разработчиков и на нем как раз и написан Bot API. Следующий способ TDLib. TDLib наиболее подходит для внедрения Telegram к себе в проект: оно инкапсулирует в себя возможности Telegram API и предоставляет понятный интерфейс, который можно изучить в подробной документации на сайте.

Также VK предоставляет все необходимые возможности для работы с зарегистрированной группой в виде REST.

Наименее доступным для разработки студентом является WhatsApp Business API, так как имеет абонентскую плату. Данная API рассчитана на большой бизнес и внедрять ее сразу же, не имея капитала – невозможно.

Из всего этого можно сделать вывод, что есть возможность реализовать общение с мессенджером через доступные интерфейсы в виде REST или написанной библиотеки TDLib.

Но для использования других мессенджеров можно прибегнуть к написанию самостоятельного приложения для получения информации без открытого API. Для этого будет использоваться Selenium WebDriver – драйвер, позволяющий моделировать поведение пользователя в браузере. Данное приложение будет неспособно обрабатывать асинхронно многие команды, например, такие как отправка сообщения, так как оно моделирует поведения пользователя в браузере. И выполнение команд может занимать довольно продолжительное время. Поэтому необходимо реализовать очередь команд, чтобы иметь возможность хранить их продолжительное время, а также сортировать в зависимости от важности. Для данной задачи хорошо подходит брокер сообщений RabbitMQ.

Для реализации проекта хорошо подходит микросервисная архитектура, так как различные части будут работать отдельно, использовать разные технологии.

Данный проект требует написания большого объема кода, связанного с серверной и клиентской частью. Для реализации серверной части потребуется код с возможностью:

- обрабатывать REST запросы;
- работать с web-драйвером Selenium;
- работать с брокером сообщений RabbitMQ;
- создавать микросервисную архитектуру.

С данными задачами справляется платформа разработки .NET, в частности ASP.NET, позволяющая создавать веб-приложения на платформе .NET.

Также к плюсам данной платформы можно отнести использование в качестве основного языка разработки – C#, который является языком со строгой типизацией и построен для использования преимущественно объектно-ориентированного подхода реализации программ.

Клиентская часть проекта должна быть выполнена на JavaScript, так как обмен между браузером клиента и сервером не должен прекращаться. Клиент должен узнавать обо всех изменениях на сервере без перезагрузки страниц.

Под данную задачу был выбран фреймворк React с использованием языка программирования TypeScript. React выбран как один из самых популярных фреймворков для написания браузерных приложений с большим количеством доступного обучающего материала. TypeScript представляет язык программирования на основе JavaScript. Причины выбора TypeScript:

- строго типизированный и компилируемый язык. На выходе компилятор создает JavaScript, который затем исполняется браузером. Однако строгая типизация уменьшает количество потенциальных ошибок, которые могли бы возникнуть при разработке на JavaScript;

- TypeScript реализует многие концепции, которые свойственны объектно-ориентированным языкам, как, например, наследование, полиморфизм, инкапсуляция и модификаторы доступа и так далее;

- потенциал TypeScript позволяет быстрее и проще писать большие сложные комплексные программы, соответственно их легче поддерживать, развивать, масштабировать и тестировать, чем на стандартном JavaScript.



## **2 Анализ данных**

Данные, с которыми работает система можно разделить на несколько групп. Входные данные – данные поступающие от пользователя в систему. Промежуточные данные – это данные, используемые системой во время работы. Выходные данные – выводимая системой информация.

В качестве входной информации в приложении выступают следующие сведения:

- данные для авторизации и аутентификации пользователя;
- данные для передачи в виде сообщений (текст, картинки, документы).

К выходным данным можно отнести:

- различные страницы для отображения диалогов с клиентами;
- данные переданные от клиентов.

### **2.1 Логическая структура данных**

Одним из преимуществ при использовании приложения является хранение всех данных непосредственно в базе данных. Для этого была разработана структура для хранения всех сообщений и их данных различных мессенджеров. Помимо сообщений база данных содержит информацию о каждом диалоге и техническую информацию.

При разработке структуры данных был применен подход Code-First. Он позволяет сначала описать структуры в виде классов, с которыми придется работать программисту, а затем уже сгенерировать таблицы в базе данных. Так как в конечном итоге в приложении работают структуры под управлением СУБД – можно представить данные в виде ER диаграммы. Описание каждой таблицы и её свойств находится в приложении А. На рисунке 2 показаны связи между структурами данных.

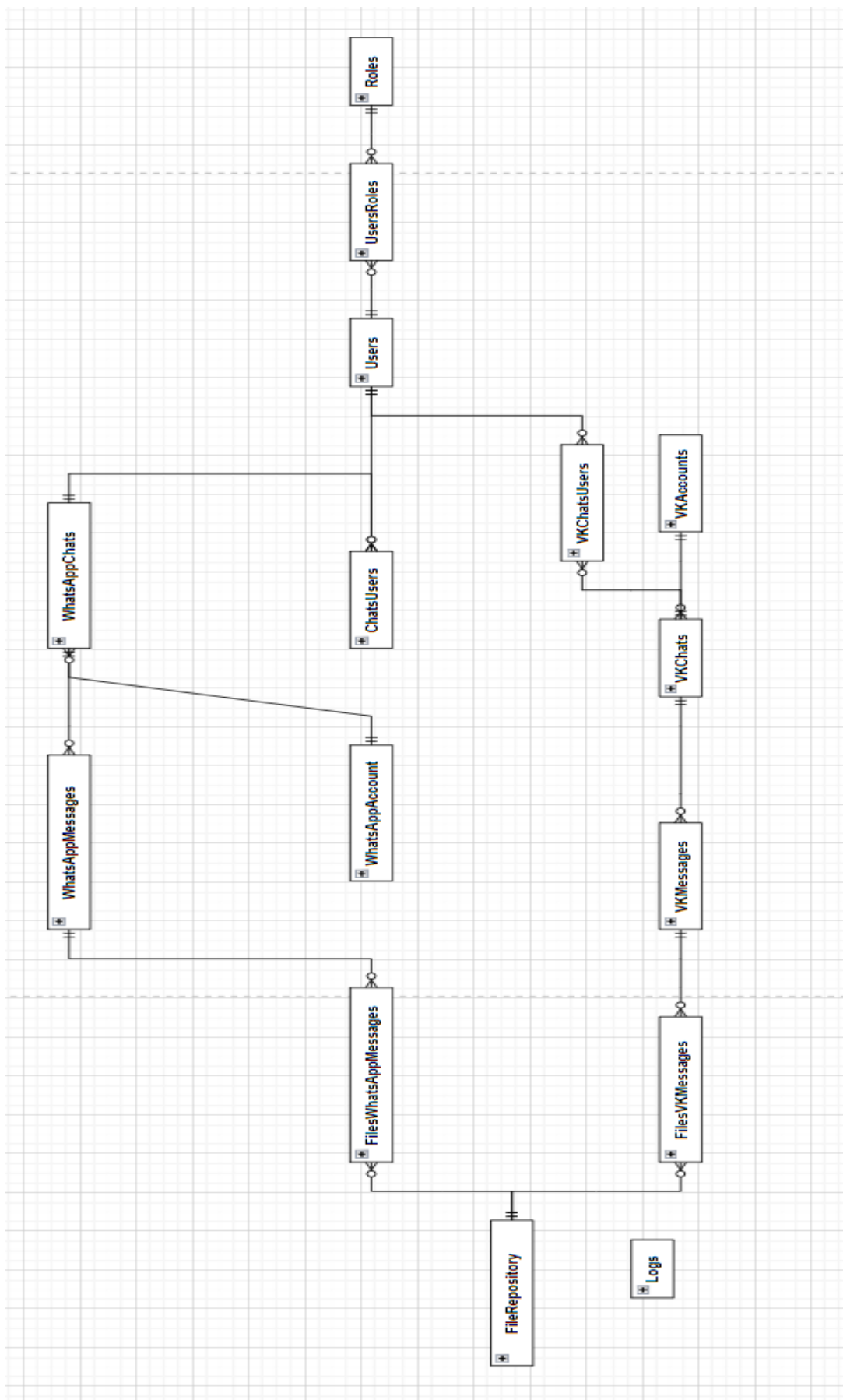


Рисунок 2 - Связи между структурами данных

### 3 Программная реализация

#### 3.1 Архитектура приложения

Для разработки приложения использовалась микросервисная архитектура. Приложение состоит из следующих сервисов:

- клиентское приложение;
- сервис для хранения управления данными приложения (ядро);
- сервис для управления WhatsApp номерами.

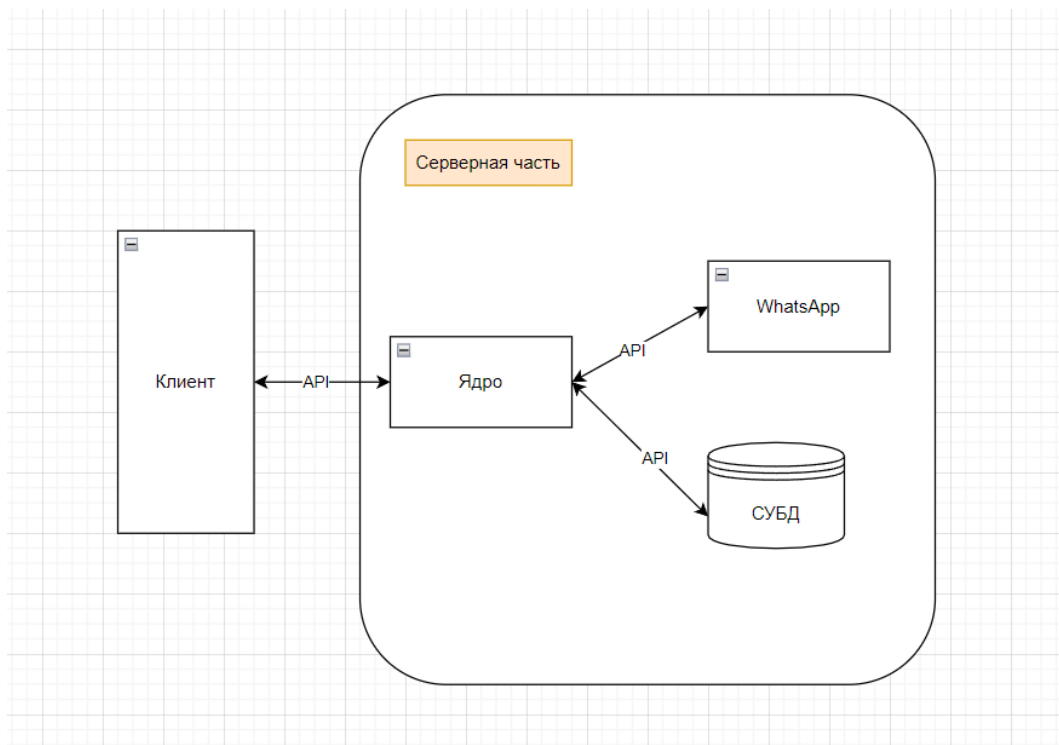


Рисунок 3 - Архитектура веб-приложения

Клиентское приложение – это программа, запускаемая в браузере пользователя, которая предоставляет удобный интерфейс для взаимодействия управления чатами.

Ядро – набор сервисов для управления, хранения и организации данных от различных мессенджеров. С ядром «общаются» все остальные сервисы. Клиентская часть получает необходимые данные только от ядра, а сервис для работы с номерами WhatsApp отправляет все необходимые данные ядру. Модуль для общения с VK API находится также в этом сервисе.

WhatsApp сервис – это приложение для обработки данных номеров WhatsApp в режиме реального времени.

СУБД – это система управления базами данных, которая хранит все необходимые данные. Описание таблиц находится в приложении А.

### **3.2 Реализация клиентской части**

Для клиентской части была выбрана Javascript-библиотека пользовательских интерфейсов React. Библиотека опирается на создании инкапсулированных компонентов с собственным состоянием и их объединении в сложные пользовательские интерфейсы.

Для клиентской части были разработаны следующие компоненты:

- ProtectedRoute – маршрутизатор, отвечающий за предоставление информации на основе jwt-токенов (ролей пользователей);
- Chat – компонент чата;
- ChatList – список чатов доступных для данного пользователя
- Message – компонент сообщения в чате;
- MessagesList – список сообщений чата;
- LoginForm – форма входа в приложение;
- InitializeCard – форма инициализации номера;
- AccountInit – компонент инициализации аккаунта;
- ChatInit – компонент инициализации отдельного чата.

### **3.3 Реализация серверной части**

Под серверной частью приложения понимается набор сервисов, работающих удаленно от компьютера пользователя на одном или нескольких серверных компьютерах. Эти сервисы следует рассмотреть отдельно.

Для реализации ядра использовалась платформа разработки веб-приложений ASP.NET. Данная платформа позволяет легко создавать REST

API для работы с данными и работать с gRPC – системой удаленного вызова процедур.

Ядро выполняет несколько похожих задач:

- общение с клиентской частью через REST API для данных, не требующих постоянного обновления;
- отправка уведомлений клиентской части в режиме онлайн;
- получение/отправка данных сервису WhatsApp;
- аутентификация и авторизация пользователей.

Для доступа к данным ядро производит проверку авторизации. Авторизация производится на основе jwt-токенов, позволяющих определить роли доступа к информации, которую предоставляет ядро.

Для решения первой задачи был написан RESTful интерфейс, позволяющий получить данные авторизованным пользователям.

Клиентская часть находится в браузере, что делает невозможным постоянный обмен информацией между сервером и клиентом без использования javascript. Поддержания постоянных обновлений в таком случае может быть реализовано тремя способами:

- websockets – протокол связи поверх TCP-соединения, предназначенный для обмена между браузером и сервером;
- server-side events – технологии отправки уведомлений сервером поверх HTTP;
- long polling – дословно «длинные опросы», постоянные опросы сервера, на предмет наличия обновления.

Для облегчения решения данной задачи была использована библиотека SignalR Core, инкапсулирующую в себе все три приведенных выше подхода.

Задачу общения с сервисом WhatsApp можно разделить на две подзадачи. Первая - отправка команд для выполнения, которые должны быть организованы в очередь и выполняться сервисом WhatsApp в том числе и по их приоритету. Вторая - получение данных от сервиса WhatsApp в асинхронном режиме.

Для решения первой подзадачи выбран программный брокер сообщений RabbitMQ, позволяющий организовывать очереди с приоритетами и отвечающий за обязательную доставку сообщения.

Для второй – система удаленного вызова процедур gRPC, использующую в качестве транспорта HTTP/2, что делает данный способ более быстрым решением нежели RESTful сервис.

### **3.4 Анализ алгоритмов**

Пример диаграммы классов, показанных на рисунке 4, описывает создание сущностей сообщений в сервисе WhatsApp для передачи данных в ядро. Сам сервис для получения данных из WhatsApp работает следующим образом: Selenium WebDriver запускает браузер и открывает страницу WhatsApp, после чего с его помощью можно эмулировать поведение пользователя в браузере; также WebDriver получает DOM-дерево, с помощью которого можно получать информацию из html документа. Для получения данных о сообщении алгоритм находит блок с классом «message», после чего, используя паттерн строитель, собирает сообщение из разных кусков. Например, в сообщении может быть текст и картинка или же присутствовать только документ или аудиосообщение.

Строитель (Builder) - шаблон проектирования, который инкапсулирует создание объекта и позволяет разделить его на различные этапы. Данный паттерн используется в следующих случаях:

- когда процесс создания нового объекта не должен зависеть от того, из каких частей этот объект состоит и как эти части связаны между собой;
- когда необходимо обеспечить получение различных вариаций объекта в процессе его создания.

Участники:

- Message – абстрактный класс сообщения;

- ConcreteMessage – пример одного из возможных типов сообщения, наследуемы от Message;
- MessageBuilder – создает объект Message;
- MessageDirector – распорядитель для создания сообщений с помощью MessageBuilder.

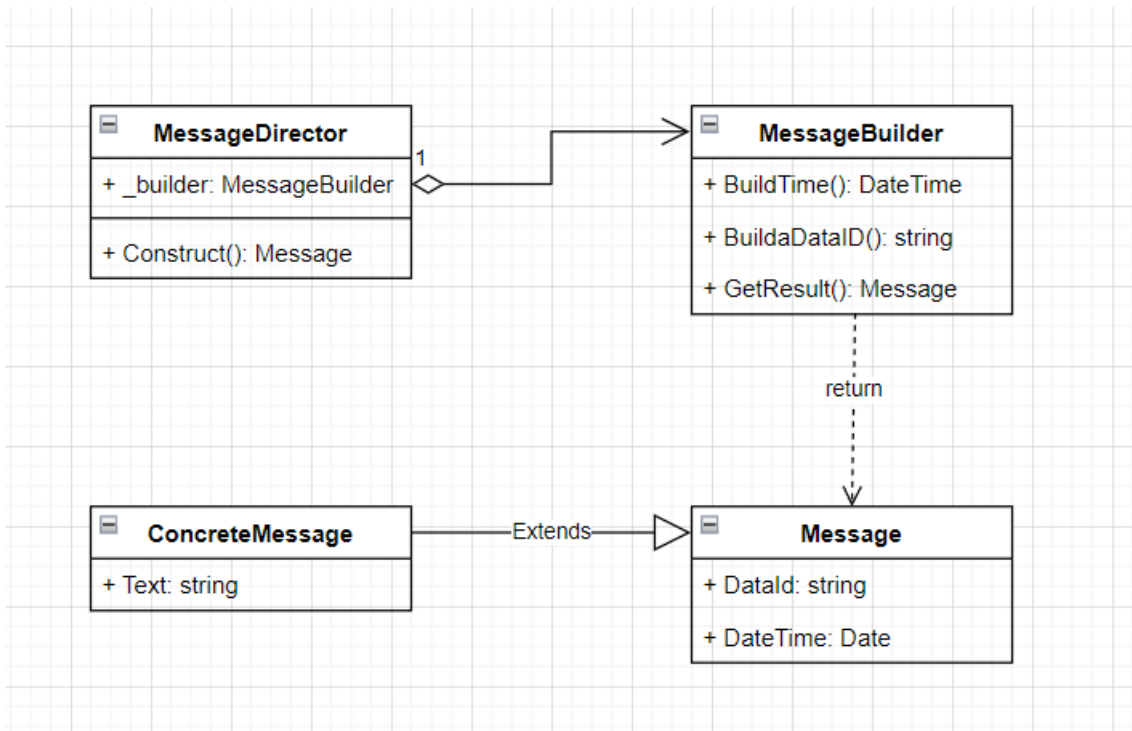


Рисунок 4 - UML диаграмма реализации паттерна строитель

Еще одним примером рабочего алгоритма может служить передача файлов по внутреннему gRPC API между сервисами. gRPC фреймворк позволяет передавать данные в виде сообщений, которые ограничены по размеру для эффективного использования памяти. В качестве максимального размера было выбрано 16384 байта. Но что если нужно передать по gRPC файл размером более указанного? Для этого используется алгоритм деления файла на «куски» и отправки их в определенном порядке на сервер, где они собираются обратно в отправленный файл. Далее пример реализации:

```

// открываем канал передачи файла
using FileStream fStream = new(path, FileMode.Open);
using var call = client.SendFileToServer();
byte[] buffer = new byte[chunkSize];
while (true)
{
    int space = chunkSize, read, offset = 0;

```

```

while (space > 0 && (read = fStream.Read(buffer, offset, space)) > 0)
{
    space -= read;
    offset += read;
}
// если буффер полный или конец файла
if (space != 0)
{
    if (offset != 0)
    {
        Array.Resize(ref buffer, offset);
        await call.RequestStream.WriteAsync(new FileChunk
        {
            Content = Google.Protobuf.ByteString.CopyFrom(buffer),
            Size = fStream.Length,
            Type = "jpg",
            Name = fileName,
        });
    }
    break;
}
else
{
    await call.RequestStream.WriteAsync(new FileChunk
    {
        Content = Google.Protobuf.ByteString.CopyFrom(buffer),
        Size = fStream.Length,
        Type = "jpg"
    });
}
}
await call.RequestStream.CompleteAsync();

```



## 4 Тестирование

В процессе разработки программного продукта параллельно производится его тестирование и отладка. Покрыть тестами весь программный продукт не представляется возможным из-за ограниченного количества времени, но основную функциональность, связанную с клиентской частью, протестировать можно с небольшими затратами ресурсов.

Клиентское приложение использует REST API для связи с ядром. Данное API можно протестировать, используя различные технологии: в процессе написания были применены Postman и Swagger UI.

Для понимания тестов уточним, что такое REST API: (также известный как RESTful API) — это интерфейс прикладного программирования (API или веб-API), который соответствует ограничениям архитектурного стиля REST и позволяет взаимодействовать с веб-службами RESTful.

REST не привязан к конкретному языку или протоколу, но наиболее часто используется посредством HTTP. Используя Postman, можно отправлять запросы на сервер для тестирования сервиса. Например, на рисунке 4 показан пример запросов для авторизации пользователя и получения jwt-токена.



Рисунок 5 - Авторизация с получением токена доступа

### Критерии стиля REST:

- **Client-Server.** Система должна быть разделена на клиентов и на серверов. Разделение интерфейсов означает, что, например, клиенты не связаны с хранением данных, которое остается внутри каждого сервера, так что мобильность кода клиента улучшается. Серверы не связаны с интерфейсом

пользователя или состоянием, так что серверы могут быть проще и масштабируемы. Серверы и клиенты могут быть заменяемы и разрабатываться независимо, пока интерфейс не изменяется.

- Stateless. Сервер не должен хранить какой-либо информации о клиентах. В запросе должна храниться вся необходимая информация для обработки запроса и если необходимо, идентификации клиента.

- Cache. Каждый ответ должен быть отмечен является ли он кэшируемым или нет, для предотвращения повторного использования клиентами устаревших или некорректных данных в ответ на дальнейшие запросы.

- Uniform Interface. Единый интерфейс определяет интерфейс между клиентами и серверами. Это упрощает и отделяет архитектуру, которая позволяет каждой части развиваться самостоятельно.

Также для быстрого документирования и просмотра работоспособности API используется Swagger UI как показано на рисунке 5.

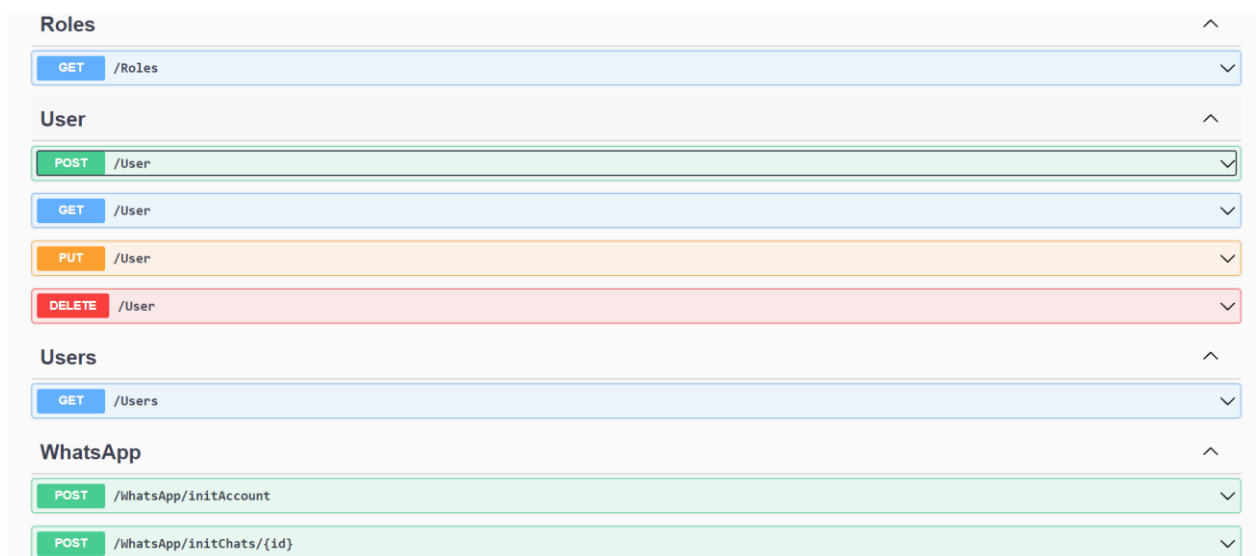


Рисунок 6 - Описание API с помощью Swagger UI

На текущий момент система находится в активной стадии разработки. При добавлении новых функций в систему производится тестирование и отладка добавленного функционала, а также производится поиск и устранение уязвимых мест в уже реализованной системе.

## **5 Техническое задание**

### **5.1 Введение**

Клиент-серверное приложение для предприятий с отделом менеджеров для общения с клиентами.

### **5.2 Назначение разработки**

Предназначен для удобного общения в одном приложении через различные мессенджеры.

### **5.3 Требования к функциональным характеристикам**

Приложение должно иметь удобный веб-интерфейс для взаимодействия с API мессенджеров или сервисом, имитирующих эти интерфейсы. В том числе предоставлять интерфейс для отправки сообщений, чтений истории сообщений чата и уведомлений о новых чатах и сообщениях.

Приложение должно предоставлять интерфейс для регистрации менеджеров и номеров различных мессенджеров.

### **5.4 Требования к надежности**

В случае возникновения ошибок программный продукт должен выводить сообщение пользователю о своей неработоспособности, чтобы пользователь мог обратиться к системному администратору для устранения проблемы.

В случае несерьезных ошибок в работе программного продукта или некорректного ввода данных пользователю должно выводиться оповещение, о том, что необходимо сделать, чтобы исправить ошибку

### **5.5 Требования к составу и параметрам технических средств**

Оптимальные требования для корректной работы сайта: необходимо иметь персональный компьютер любой операционной системой, с оперативной памятью не менее 512 Мб, также свободного места на диске должно быть не менее 700 Мб, процессор с частотой 600 МГц.

### **5.6 Требования к программной документации**

Программная документация должна содержать руководство пользователя.

### **5.7 Требования к информационной, программной совместимости**

Сервер не требует определённой операционной системы. На рабочей станции должен быть установлен любой браузер.

## 6 Руководство пользователя

При запуске приложения необходимо авторизоваться для дальнейших действий.

Рисунок 7 - Форма авторизации

После чего администратор может зарегистрировать новые аккаунты для менеджеров или изменить существующие.

Рисунок 8 - Форма для добавления новых аккаунтов

На форме инициализации номеров можно подключить новые номера.

Logo

UserManagment ChatPage

Номер телефона

Включить аккаунт

Зарегистрированные аккаунты:  
Номер аккаунта: 89244562334  
Состояние инициализации: Отсканируйте qr код




Рисунок 9 - Форма добавления номеров WhatsApp

Для инициализации номеров WhatsApp можно выбрать какие именно чаты необходимо подключить.

+7 914 144-45-01

Прокручиваем чат вверх. Текущая дата прокрутки: 22.02.2022, 03:00

Рисунок 10 - Пример инициализации чата

После чего данный чат будет доступен для использования.

Здравствуй

14.03.2022 17:28:02

123

16.03.2022 16:27:00

asd

16.03.2022 16:31:00

Admin

Manager

Авторизация

Зарегистрировать менеджеров

Просмотреть статистику

Подключить новый номер

Авторизация

Просмотреть чаты

Просмотр истории конкретного чата

Написать сообщение в конкретном чате

Написать первым

Document

Picture

Введите сообщение...

Отправить

Рисунок 11 - Пример чата

## ЗАКЛЮЧЕНИЕ

В ходе проделанной работы было разработано клиент-серверное приложение для общения с клиентами посредством мессенджеров.

Данное приложение позволит общаться с клиентами из одного окна и делать это с нескольких устройств. Также вся информация из чатов будет храниться непосредственно на сервере компании, что позволит обращаться даже к старым данным.

В дальнейшем проект будет развиваться и дорабатываться. Следующими задачами будут добавление, новых мессенджеров, добавление CRM систем для бизнеса и улучшение интерфейса.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Учебник по ASP.NET [Электронный ресурс]. - <https://metanit.com/sharp/aspnet5/1.2.php> (дата обращения 20.02.2022)
2. Документация по веб-драйверу Selenium [Электронный ресурс]. - <https://www.selenium.dev/documentation/> (дата обращения 21.02.2022)
3. Документация фреймворка gRPC [Электронный ресурс]. - <https://grpc.io/docs/> (дата обращения 24.02.2022)
4. Руководство RabbitMQ [Электронный ресурс]. - <https://www.rabbitmq.com/getstarted.html> (дата обращения 21.02.2022)



## ПРИЛОЖЕНИЕ А

(обязательное)

### Описание таблиц базы данных

Таблица А.1 – Таблицы базы данных

Название таблицы	Описание
Roles	Хранит роли пользователей
Users	Информация о пользователях
WhatsAppChats	Информация о чатах whatsapp
WhatsAppMessages	Информация о сообщениях whatsapp
WhatsAppAccounts	Информация об аккаунтах whatsapp
VKChats	Информация о чатах VK
VKMessages	Информация о сообщениях VK
VKAccounts	Информация об аккаунтах VK
FileRepository	Информация о сохраненных файлах
Logs	Логи системы

Таблица А.2 – Описание таблицы Roles

Название	Описание	Тип	Память
ID	Первичный ключ	int	4 байта
Name	Название роли	string	4 байта

Таблица А.3 – Описание таблицы Users

Название	Описание	Тип	Память
ID	Первичный ключ	Int	4 байта
Email	Email пользователя	String	4 байта
Password	Пароль	String	4 байта

Таблица А.4 – Описание таблицы WhatsAppChats

Название	Описание	Тип	Память
ID	Первичный ключ	Int	4 байта
Title	Заголовок чата	String	4 байта
InitStatus	Статус инициализации	Byte	1 byte
InitDate	Дата последнего инициализированного сообщения	Int	4 байта

Таблица А.5 – Описание таблицы WhatsAppMessages

Название	Описание	Тип	Память
ID	Первичный ключ	Int	4 байта
DataId	Уникальный идентификатор в приложении WhatsApp	String	4 байта
Date	Дата	Int	4 байта
IsOutgoing	Исходящее ли сообщение	Bool	1 байт
IsReaded	Прочитано ли пользователем	Bool	1 байт
ChatId	Внешний ключ WhatsAppChats	Int	4 байта
Text	Текст сообщения	String	4 байта
DocumentName	Название документа прикрепленного к сообщению	String	4 байта

Таблица А.6 – Описание таблицы WhatsAppAccounts

Название	Описание	Тип	Память
ID	Первичный ключ	Int	4 байта
Number	Номер аккаунта	String	4 байта
InitState	Статус инициализации аккаунта	Byte	1 байт

Таблица А.7 – Описание таблицы VKChats

Название	Описание	Тип	Память
ID	Первичный ключ	Int	4 байта
VKID	Идентификатор в БД VK	Int	4 байта
Title	Заголовок чата	String	4 байта

Таблица А.8 – Описание таблицы VKMessages

Название	Описание	Тип	Память
ID	Первичный ключ	Int	4 байта
VKID	Идентификатор в БД VK	Int	4 байта
Text	Текст сообщения	String	4 байта

Таблица А.9 – Описание таблицы VKAccounts

Название	Описание	Тип	Память
ID	Первичный ключ	Int	4 байта
Token	Уникальный токен группы	String	4 байта

Таблица А.10 – Описание таблицы FileRepository

Название	Описание	Тип	Память
ID	Первичный ключ	Int	4 байта
Hash	Хэш файла (sha256)	String	4 байта
Name	Название файла в хранилище	String	4 байта
OrdinaryName	Обычное название файла вместе с расширением	String	4 байта

Таблица А.11 – Описание таблицы Logs

Название	Описание	Тип	Память
ID	Первичный ключ	Int	4 байта
Text	Текст лога	String	4 байта
Date	Дата	Int	4 байта