МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

**КУРСОВОЙ ПРОЕКТ**

По дисциплине: «Системное программирование»

на тему: «Брокер сообщений в экосистеме Linux, модуль “Хранение сообщений”»

|  |  |  |
| --- | --- | --- |
| Исполнитель: |  | студентка группы 10701322  Ю.В. Махонина |
| Преподаватель: |  | Азаревич В.А. |

Минск 2025

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту**

По дисциплине: «Системное программирование»

на тему: «Брокер сообщений в экосистеме Linux, модуль “Хранение сообщений”»

|  |  |  |
| --- | --- | --- |
| Исполнитель: |  | студентка группы 10701322  Ю.В. Махонина |
| Преподаватель: |  | Азаревич В.А. |

Минск 2025

**ОГЛАВЛЕНИЕ**

[**ВВЕДЕНИЕ** 4](#_Toc193571552)

[1 Описание и анализ предметной области. Обзор аналогов 5](#_Toc193571553)

[1.1 Описание и анализ предметной области 5](#_Toc193571554)

[1.2 Обзор аналогов 6](#_Toc193571555)

[2 Требования к программному обеспечению 8](#_Toc193571556)

[2.1 Требования к системе 8](#_Toc193571557)

[3 Изучение доступных библиотек для решения поставленной задачи. Ознакомление с документацией и примерами использования выбранной библиотеки. 10](#_Toc193571558)

[3.1 JDBC 10](#_Toc193571559)

[3.2 JavaFX 12](#_Toc193571560)

[4 Проектирование программного обеспечения 14](#_Toc193571561)

[4.1 Диаграммы вариантов использования 14](#_Toc193571562)

[4.2 Проектирование базы данных 15](#_Toc193571563)

[4.3 Архитектура программного обеспечения 17](#_Toc193571564)

[4.4 Проектирование основных классов 18](#_Toc193571565)

[5 Реализация программного обеспечения 19](#_Toc193571566)

[5.1 Реализация серверной части 19](#_Toc193571567)

[5.3 Система контроля версий, удаленный репозиторий хранения версий разрабатываемой программы. Стратегия разработки. 19](#_Toc193571568)

[5.4 Безопасность. Шифрование данных. Защита от SQL-инъекций и других уязвимостей. 19](#_Toc193571569)

[5.5 Механизмы межпроцессного взаимодействия (проецирование файлов, разделяемая память, сокеты, очереди сообщений, сигналы). 19](#_Toc193571570)

[5.6 Многозадачность и многопоточность, а также средства их синхронизации (семафоры, мьютексы). 19](#_Toc193571571)

[5.7 Модуль приложения 19](#_Toc193571572)

[5.8 Пользовательские библиотеки 19](#_Toc193571573)

[5.9 Жерналирование 19](#_Toc193571574)

[5.10 Развертывание приложения, создание установочного пакета 19](#_Toc193571575)

[6 Тестирование программного обеспечения 20](#_Toc193571576)

[6.1 Тесты для проверки функциональности системы 20](#_Toc193571577)

[6.2 Тестирование на различных конфигурациях и сценариях использования. 20](#_Toc193571578)

[7 Документация 21](#_Toc193571579)

[7.1 Документация по установке и использованию приложения. 21](#_Toc193571580)

[7.2 Описание структуры кода и основные функции 21](#_Toc193571581)

[**ЗАКЛЮЧЕНИЕ** 22](#_Toc193571582)

[**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ** 23](#_Toc193571583)

[ПРИЛОЖЕНИЕ А 24](#_Toc193571584)

## **ВВЕДЕНИЕ**

В современном цифровом мире обмен информацией между программными компонентами играет ключевую роль в обеспечении стабильной работы и быстрого реагирования на события. С увеличением объёма передаваемых данных и усложнением архитектуры программных систем растёт потребность в надёжных и масштабируемых средствах обмена сообщениями. Одним из решений является разработка брокера сообщений – системы, которая обеспечивает эффективное и безопасное взаимодействие между различными сервисами и приложениями.

Разработанная система «Брокер сообщений в экосистеме Linux» ориентирована на создание надёжного инструмента для передачи, обработки и хранения сообщений как внутри распределённой системы, так и при интеграции с внешними сервисами. В рамках данного проекта особое внимание уделено модулю «Хранение сообщений», который отвечает за долговременную и безопасную фиксацию передаваемых данных. Реализация этого модуля предусматривает использование языка программирования Java для построения логики приложения и SQL-базы данных для организации системы хранения, что обеспечивает высокую производительность, масштабируемость и совместимость с существующими решениями в Unix-подобных системах.

Актуальность разработки данного модуля определяется увеличением спроса на системы обмена сообщениями, способные работать в режиме реального времени, а также требовательностью к сохранности и целостности данных. Накопление сообщений, их маршрутизация и последующая обработка требуют надёжного хранения, которое должно обеспечивать быстрый доступ к информации с возможностью последующего анализа и аудита. Кроме того, использование экосистемы Linux предоставляет преимущества в виде стабильности, безопасности и эффективности при работе с серверными приложениями.

Основными задачами, решаемыми в рамках разработки модуля «Хранение сообщений», являются:

1. Построение структурированной и масштабируемой базы данных, способной обрабатывать большой объём информации;
2. Обеспечение целостности и безопасности данных посредством реализации механизма контроля доступа и использования внешних ключей;
3. Оптимизация работы системы через применение современных методов кэширования, многоуровневой архитектуры и транзакционной обработки операций;
4. Интеграция с другими компонентами брокера сообщений для создания единой архитектуры, позволяющей обеспечить стабильное взаимодействие между модулями.

## Описание и анализ предметной области. Обзор аналогов

В условиях современного развития информационных технологий распределённые системы и микросервисная архитектура становятся стандартом построения корпоративных приложений. Одной из ключевых составляющих таких систем является механизм обмена сообщениями между различными компонентами, что позволяет обеспечить гибкость, масштабируемость и отказоустойчивость. Именно в этой связи разработка брокера сообщений приобретает особую актуальность. Брокер сообщений представляет собой программное обеспечение, которое принимает, обрабатывает, сохраняет и пересылает сообщения между отправителями и получателями, обеспечивая асинхронное взаимодействие компонентов системы.

В экосистеме Linux данная технология имеет важное значение за счёт высокой стабильности, безопасности и широкого набора инструментов для разработки распределённых приложений. Модуль «Хранение сообщений», являющийся составной частью брокера, отвечает за долговременное и структурированное хранение информации, поступающей от различных источников. Он обеспечивает сохранность данных, контроль целостности сообщений, а также быстрый доступ к записям для последующей обработки. Модуль должен учитывать большие объёмы данных, гарантировать транзакционную целостность и поддерживать высокую скорость операций чтения и записи.

### 1.1 Описание и анализ предметной области

Одним из основных аспектов анализа предметной области является необходимость обработки сообщений в режиме реального времени с возможностью проведения последующего аудита. В современных системах обмена информацией важно не только доставлять сообщение от отправителя к получателю, но и сохранять временные метки, статусы обработки, а также логи с информацией о возникших ошибках или задержках. Это позволяет анализировать работу системы, оптимизировать маршруты передачи данных и проводить диагностику проблем. Таким образом, модуль хранения сообщений влияет на общую надёжность и масштабируемость всей инфраструктуры брокера, что требует тщательного подхода к проектированию базы данных и архитектуры приложения.

Другим важным направлением анализа является согласованность и отказоустойчивость системы. При работе с большими объёмами сообщений возможны сбои и потери данных, поэтому крайне важно предусмотреть механизмы восстановления и обработки ошибок. Использование транзакций в СУБД, репликация данных и разработка механизмов резервного копирования позволяют обеспечить высокую степень защиты информации и минимизировать время простоя системы. Эти особенности особенно актуальны для систем, функционирующих в условиях высокой нагрузочной интенсивности, например, в финансовых системах или онлайн-платформах для электронной коммерции.

Немаловажным аспектом является и безопасность хранения сообщений. Наряду с обеспечением целостности данных должна быть реализована комплексная система контроля доступа, включающая аутентификацию пользователей и разграничение прав для различных слоёв системы. В современных реализациях брокеров сообщений используется шифрование данных при передаче и хранении, что позволяет защитить информацию от несанкционированного доступа и хакерских атак. Это становится особенно актуально в условиях растущего числа кибератак и угроз информационной безопасности.

Эффективная обработка сообщений во многом зависит от архитектурных решений, реализуемых в модуле хранения. Применение реляционной базы данных, такой как MySQL позволяет использовать преимущества транзакционной модели, в то время как комбинация с NoSQL-системами даёт возможность организации горизонтального масштабирования. В перспективе возможна интеграция с системами распределённого кэширования, что дополнительно ускоряет операции выборки данных. Таким образом, анализ предметной области показывает, что разработка модульного решения для хранения сообщений должна учитывать множество факторов – от скорости обработки и экономии ресурсов до безопасности и отказоустойчивости.

### 1.2 Обзор аналогов

При сравнительном анализе аналогов ключевыми критериями являются производительность, масштабируемость, удобство интеграции и наличие механизмов мониторинга и безопасности. В контексте разрабатываемой системы для Linux, модуль «Хранение сообщений» может быть выполнен с использованием традиционной SQL-базы данных, где упор делается на транзакционную целостность, поддержку внешних ключей и возможность аудита. Такой подход обеспечивает высокую надежность и предсказуемость поведения системы, что особенно важно при работе с критически важными данными. Кроме того, использование проверенных технологий, таких как MySQL или PostgreSQL, позволяет воспользоваться обширной документацией и сообществом разработчиков, что ускоряет процесс внедрения и эксплуатации системы.

Одной из важных задач при выборе архитектурного решения является обеспечение совместимости с уже существующими системами и компонентами. В ряде случаев может оказаться целесообразным выстроить систему хранения сообщений в виде микросервиса, который отдельно отвечает за долговременное хранение и обратную связь с брокером сообщений. Такой подход позволяет масштабировать систему по мере роста объема передаваемой информации, а также внедрить дополнительные механизмы анализа, мониторинга и восстановления данных. Анализ аналогов показывает, что интеграция с облачными решениями и использование кластерных технологий становится актуальной тенденцией, позволяющей достигать высокой отказоустойчивости и производительности.

* + 1. Apache Kafka

На современном рынке существует множество решений для обмена сообщениями и хранения данных, каждое из которых обладает своими особенностями, преимуществами и недостатками. Одним из наиболее известных аналогов является Apache Kafka. Это распределённая платформа для стриминговой обработки данных, которая позволяет обрабатывать и хранить огромные объёмы сообщений в режиме реального времени. Kafka обладает высокой производительностью, масштабируемостью и надежностью благодаря горизонтальному масштабированию и возможности репликации данных. Однако, её сложная настройка и необходимость глубокого понимания принципов распределённых систем требуют определённых знаний и опыта.

1.2.2 RabbitMQ

Еще одно популярное решение RabbitMQ – брокер сообщений, реализованный по стандарту AMQP. RabbitMQ применяет подход очередей, что обеспечивает упорядоченную доставку сообщений. Он легко настраивается, поддерживает множество протоколов обмена данными и может быть интегрирован с разными технологиями. К недостаткам можно отнести сравнительно меньшую скорость обработки при очень высокой нагрузке, что может стать ограничивающим фактором в крупных системах.

1.2.3 ActiveMQ

Еще одним аналогом служит ActiveMQ – брокер сообщений, разработанный с упором на простоту интеграции с Java-системами. ActiveMQ широко используется в корпоративных приложениях благодаря своей гибкости в настройке, возможности работы с различными протоколами и поддержке кластеризации. Его архитектура позволяет использовать как традиционные реляционные базы данных, так и NoSQL-хранилища для долговременного хранения сообщений. Несмотря на это, при экстремальных нагрузках ActiveMQ показывает меньшую производительность по сравнению с Kafka, что требует адаптации конфигурации при масштабировании.

## 2 Требования к программному обеспечению

Проект «Брокер сообщений в экосистеме Linux» направлен на создание надёжного и масштабируемого решения для организации обмена информацией между сервисами и компонентами информационных систем. Модуль «Хранение сообщений» является критически важной составляющей общей системы, так как отвечает за долговременное и целостное сохранение, обработку и последующую выборку сообщений, поступающих от различных источников.

Целью данного модуля является обеспечение:

1. Стабильного и корректного переноса сообщений из входящих потоков в хранилище с гарантией сохранности данных;
2. Возможность анализа истории сообщений, их статусов и временных меток для диагностики и аудита системы;

### 2.1 Требования к системе

2.1.1 Типы пользователей

Для обеспечения корректной работы модуля «Хранение сообщений» предполагается взаимодействие следующих групп пользователей:

* Системный администратор

Роль: Управление настройками модуля, мониторинг работы системы, обновление конфигурации базы данных и обеспечение безопасности доступа.

Задачи: Настройка подключения к SQL-серверу, резервное копирование данных и логирование событий для последующего аудита.

* Пользователь

Роль: Получение информации о статусе сообщений, осуществление запросов к модулю для проверки истории обработки.

Задачи: Использование предоставленных интерфейсов для просмотра отчётов, анализа передачи данных и получения актуальной информации о сообщениях.

2.1.2 Функциональные требования

* Приём и валидация сообщений

Модуль должен принимать входящие сообщения, передаваемые через сетевые протоколы (WebSocket). Предусмотреть валидацию сообщений по ключевым параметрам (наличие обязательных полей, корректность структурированной информации, статусов и валидность временных меток).

* Хранение сообщений в базе данных

Запись сообщений с указанием всех атрибутов: идентификатор сообщения, информация об отправителе и получателе, статус, время создания и обновления. Обеспечение целостности данных с помощью транзакций и внешних ключей. Поддержка истории изменений статуса сообщений (с возможностью аудита и восстановления информации).

* Обеспечение доступа к данным

Реализовать API для выборки сообщений по различным критериям: временной диапазон, определённый статус, отправитель или получатель. Предусмотреть возможность сортировки и фильтрации данных для ускорения обработки запросов.

* Обработка ошибок и логирование

В случае возникновения ошибок при записи или обновлении данных система должна регистрировать подробные сообщения об ошибках (лог).

* Интеграция с модулем маршрутизации

После успешного сохранения сообщения модуль должен передавать информацию в компонент маршрутизации, который занимается дальнейшей доставкой. Обеспечить синхронизацию состояния сообщения между модулями посредством обновления статуса.

* Администрирование и отчётность

Обеспечить интерфейсы для просмотра текущего состояния системы, отчётов по количеству обработанных сообщений, ошибок и статистики по времени обработки. Реализовать возможность ручного вмешательства (изменение статусов, повторная обработка сообщений) через административные инструменты.

* + 1. Нефункциональные требования
* Производительность

Модуль должен обрабатывать более 10 сообщений в режиме реального времени с минимальной задержкой. Время обработки и выборки сообщений не должно превышать установленого порогового значения в 500 миллисекунд.

* Надёжность и отказоустойчивость

Система должна гарантировать 100% целостность данных при любом виде операций, используя механизмы транзакций и резервного копирования. Обработка ошибок должна быть автоматизирована: в случае сбоев система должна восстанавливать соединения, проводить повторные попытки записи и своевременно оповещать оператора о критических ошибках.

* Безопасность

Обеспечить механизм аутентификации и авторизации пользователей, взаимодействующих с модулем, чтобы предотвратить несанкционированный доступ. Реализовать шифрование данных при передаче и хранении, обеспечить соответствие стандартам безопасности в информационных системах.

* Удобство использования

Предусмотреть интуитивно понятный административный интерфейс (или консольное приложение) для мониторинга состояния системы, управления сообщениями и подготовки отчётов.

## 3 Изучение доступных библиотек для решения поставленной задачи. Ознакомление с документацией и примерами использования выбранной библиотеки.

При реализации модуля «Хранение сообщений» ключевыми аспектами являются работа с базой данных и обеспечение удобного пользовательского интерфейса для мониторинга, конфигурации и отладки функциональности. В рамках данной задачи особый интерес представляют две библиотеки: JDBC и JavaFX.

### 3.1 JDBC

Java Database Connectivity (JDBC) является стандартным API для доступа к реляционным базам данных из приложений на Java. Его применение оправдано в случаях, когда требуется эффективное взаимодействие с базой данных, выполнение SQL-запросов, управление транзакциями и обработка результатов запросов. В контексте модуля «Хранение сообщений» использование JDBC позволяет непосредственно обращаться к таблицам для сохранения, выборки и обновления информации.

JDBC предоставляет достаточно низкоуровневый доступ к данным, что даёт разработчику полный контроль над процессом выполнения запросов. При этом понятие "слабой связности" между приложением и базой данных позволяет гибко адаптировать систему под изменяющиеся бизнес-требования, оптимизировать запросы и повышать производительность при необходимости.

Oracle и другие поставщики СУБД (например, MySQL, PostgreSQL) предоставляют обширную документацию по JDBC. Документация голосит о следующих ключевых понятиях:

* Connection: Объект, который устанавливает связь с базой данных через JDBC-драйвер.
* Statement/PreparedStatement: Объекты для выполнения SQL-запросов. PreparedStatement особенно полезен, когда требуется выполнять один и тот же запрос с разными параметрами для предотвращения SQL-инъекций.
* ResultSet: Представление возвращаемых данных запросов. Оно позволяет последовательно перебрать строки результата.

Пример ниже демонстрирует, как установить соединение с базой данных, создать объект Statement, выполнить запрос и обработать результаты. Код взят из общих примеров, представленных в документации JDBC.

* DriverManager.getConnection() используется для установления соединения с базой данных.
* Statement позволяет выполнять статические SQL-запросы.
* ResultSet возвращает результаты запроса, после чего мы их обходим с помощью метода next().
* Рекомендуется использовать блок finally для закрытия всех ресурсов (или применять конструкцию try-with-resources, если СУБД и версия Java это поддерживают).

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class JdbcExample {

public static void main(String[] args) {

// URL подключения зависит от используемой СУБД.

// В примере используется MySQL. Замените значения на свои данные.

String url = "jdbc:mysql://localhost:3306/testdb";

String user = "username";

String password = "password";

Connection connection = null;

Statement statement = null;

ResultSet resultSet = null;

try {

// Устанавливаем соединение

connection = DriverManager.getConnection(url, user, password);

// Создаем объект Statement для выполнения запроса

statement = connection.createStatement();

// Выполняем запрос к базе данных

resultSet = statement.executeQuery("SELECT id, name FROM mytable");

// Обрабатываем результаты запроса

while (resultSet.next()) {

int id = resultSet.getInt("id");

String name = resultSet.getString("name");

System.out.println("ID: " + id + ", Name: " + name);

}

} catch (SQLException ex) {

ex.printStackTrace();

} finally {

// Закрываем все ресурсы в finally-блоке для корректного освобождения памяти

try { if (resultSet != null) resultSet.close(); } catch (SQLException e) { e.printStackTrace(); }

try { if (statement != null) statement.close(); } catch (SQLException e) { e.printStackTrace(); }

try { if (connection != null) connection.close(); } catch (SQLException e) { e.printStackTrace(); }

}

}

}

### 3.2 JavaFX

JavaFX – это современная платформа для создания графических пользовательских интерфейсов (GUI) на Java. Эта библиотека отлично подходит для разработки приложений с богатым функционалом, привлекательным дизайном и высокой интерактивностью. В контексте модуля «Хранение сообщений» JavaFX может использоваться для визуализации данных, мониторинга системы, а также для создания административных панелей, где можно просматривать аудиторские записи, логи ошибок и статус сообщений.

JavaFX имеет обширную документацию на официальном сайте Oracle, где подробно описаны основы работы с библиотекой, компоненты пользовательского интерфейса, а также механизмы связывания данных. Основные элементы, на которые стоит обратить внимание:

* Stage и Scene: Основные контейнеры, которые представляют окно приложения и его содержимое.
* FXML: XML-язык для описания интерфейсов, который позволяет отделить логику от представления. FXML помогает быстро прототипировать и изменять внешний вид интерфейса без необходимости перекомпиляции кода.
* Контролы и компоненты: Таблицы, формы, кнопки, поля ввода и прочие элементы, позволяющие реализовать функциональность администрирования и мониторинга.

Пример ниже демонстрирует создание простого окна с надписью. Пример взят из официальной документации и учебных руководств по JavaFX.

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.Label;

import javafx.scene.layout.StackPane;

import javafx.stage.Stage;

public class HelloJavaFX extends Application {

@Override

public void start(Stage primaryStage) {

// Создаем элемент управления Label с приветственным текстом

Label helloLabel = new Label("Hello, JavaFX!");

// Используем StackPane как корневой элемент сцены

StackPane root = new StackPane();

root.getChildren().add(helloLabel);

// Создаем сцену, задавая размеры окна

Scene scene = new Scene(root, 300, 200);

// Настраиваем окно приложения (Stage)

primaryStage.setTitle("Пример JavaFX");

primaryStage.setScene(scene);

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

* Метод start() является точкой входа для всех JavaFX-приложений.
* Stage представляет главное окно приложения, а Scene содержит графические элементы.
* StackPane используется для простого позиционирования элементов по центру.

## Проектирование программного обеспечения

Проектирование – один из важных шагов при разработке программы, который очень часто игнорируется начинающими разработчиками. Обычно они пытаются удержать всё в голове или, в лучшем случае, записать некоторые важные сведения на листе бумаги. Как результат, у них нет чёткого плана дальнейших действий, и проект может быть отложен в долгий ящик.

Обычно при проектировании разработчики изображают систему графически, поскольку человеку легко разобраться в таком представлении. Именно поэтому вместо написания громоздких текстов про каждую возможность будущей программы разработчики строят различные диаграммы и таблицы для описания своих систем. Это помогает не забывать, что нужно реализовать в программе, и быстро вводить в курс дела своих коллег.

### Диаграммы вариантов использования

Диаграмма вариантов использования (англ. use-case diagram) – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей.

Диаграмма вариантов использования — это одна из разновидностей диаграммы UML, которая используется для описания возможного сценария использования разрабатываемой системы. Иными словами, она демонстрирует, как потенциальный пользователь будет взаимодействовать с системой.

На рисунке 4.1 представлена диаграмма вариантов использования для разрабатываемой информационной системы. Здесь выделено три основных сущности: незарегистрированный пользователь, зарегистрированный пользователь и администратор.

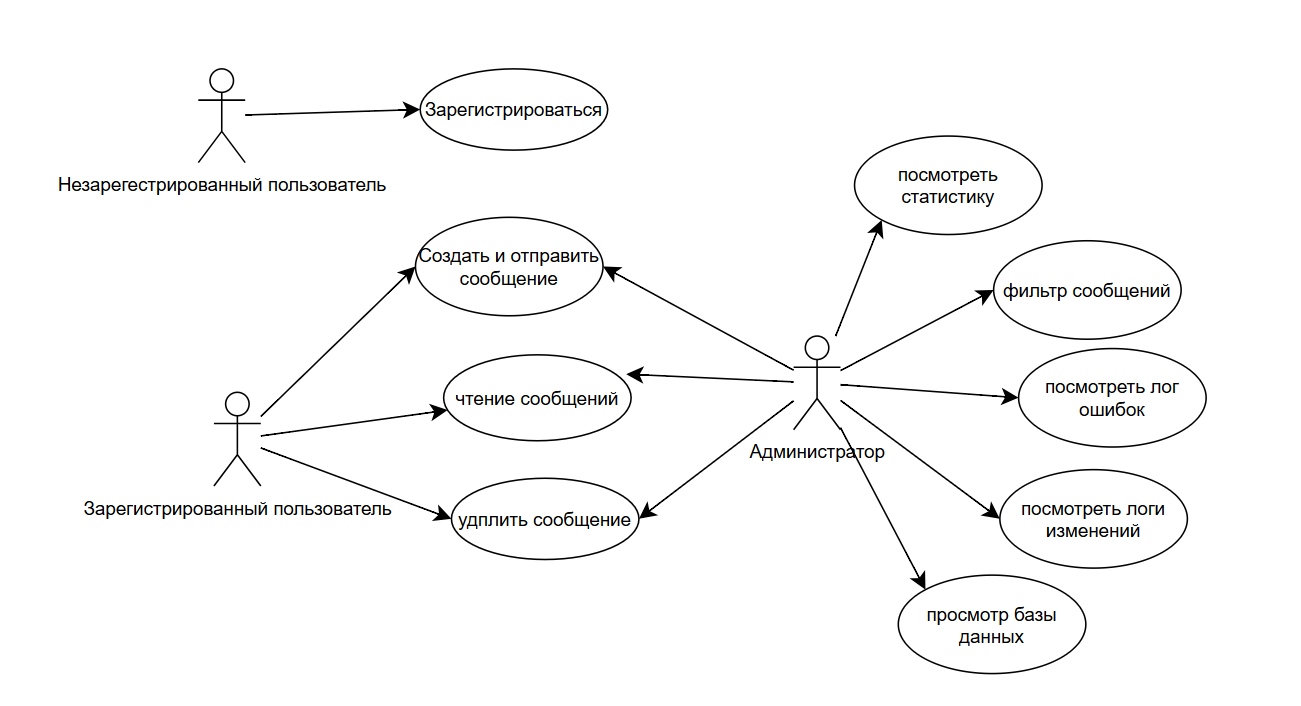


Рисунок 4.1 – Диаграмма вариантов использования

### Проектирование базы данных

В данном разделе описывается проектирование базы данных для информационной системы. В процессе проектирования определяются основные сущности и их взаимосвязи, устанавливаются ограничения для обеспечения целостности и достоверности информации, а также разрабатывается схема, оптимизированная для быстрого доступа и обработки запросов.

Опишем основные таблицы базы данных под названием MessagesDB.

Таблица Users (таблица 1) хранит информацию о пользователях системы. Каждая запись включает уникальный идентификатор, имя, пароль и email, что позволяет идентифицировать пользователей и контролировать их доступ.

Таблица 1 – таблица Users

|  |  |  |  |
| --- | --- | --- | --- |
| Столбец | Тип данных | Ограничения | Описание |
| UserID | INT | PRIMARY KEY, AUTO\_INCREMENT | Уникальный идентификатор пользователя |
| Name | VARCHAR(20) | NOT NULL, CHECK (CHAR\_LENGTH(name) BETWEEN 3 AND 20) | Имя пользователя (от 3 до 20 символов) |
| Password | VARCHAR(20) | NOT NULL, CHECK (CHAR\_LENGTH(name) BETWEEN 6 AND 20) | Хэш пароля пользователя (от 6 до 20 символов) |
| email | VARCHAR(40) | NOT NULL, CHECK (email LIKE ‘%@%’) | Электронная почта (должно содержать символ «@») |
| isAdmin | BOOLEAN | NOT NULL, DEFAULT FALSE | Флаг администратора |

Таблица Messages (таблица 2) содержит данные о сообщениях, отправленных между пользователями. Таблица фиксирует отправителя, получателя, текст сообщения, его статус и временные метки создания и обновления, что обеспечивает отслеживание потока информации.

Таблица 2 – таблица Messages

|  |  |  |  |
| --- | --- | --- | --- |
| Столбец | Тип данных | Ограничения | Описание |
| MessageID | INT | PRIMARY KEY, AUTO\_INCREMENT | Уникальный идентификатор сообщения |
| SenderID | INT | NOT NULL, FOREIGN KEY (ссылается на Users.UserID) | Идентификатор отправителя сообщения |
| ReceiverID | INT | NOT NULL, FOREIGN KEY (ссылается на Users.UserID) | Идентификатор получателя сообщения |
| textMessage | TEXT |  | Текст сообщения |
| Status | ENUM (‘new’, ‘error’, ‘sent’, ‘received’) | NOT NULL | Статус сообщения |
| timeCreating | DATETIME | DEFAULT CURRENT\_TIMESTAMP | Время создания сообщения, по умолчанию устанавливается текущее время |
| timeUpdating | DATETIME | DEFAULT CURRENT\_TIMESTAMP  ON UPDATE  CURRENT\_TIMESTAMP | Время последнего обновления сообщения, обновляется при обновлении записи |

Таблица MessageAudit (таблица 3) предназначена для логирования изменений в статусе сообщений. Здесь фиксируются изменения (старый и новый статус), время их совершения и комментарии, что необходимо для аудита и анализа истории изменения данных.

Таблица 3 – таблица MessageAudit

|  |  |  |  |
| --- | --- | --- | --- |
| Столбец | Тип данных | Ограничения | Описание |
| AuditID | INT | PRIMARY KEY, AUTO\_INCREMENT | Уникальный идентификатор записи аудита |
| MessageID | INT | FOREIGN KEY (ссылается на Messages.MessageID) | Идентификатор сообщения, к которому относится запись |
| OldStatus | ENUM (‘new’, ‘error’, ‘sent’, ‘received’) | NOT NULL | Прежний статут сообщения |
| NewStatus | ENUM (‘new’, ‘error’, ‘sent’, ‘received’) | NOT NULL | Новый статус сообщения |
| ChangeAt | DATETIME | DEFAULT CURRENT\_TIMESTAMP | Время изменения статуса, по умолчанию текущее время |
| Comment | TEXT |  | Комментарий к изменению |

Таблица ErrorLog (таблица 4) регистрирует ошибки, возникающие в процессе работы с сообщениями. Каждая запись включает описание ошибки, привязку к сообщению (если применимо) и время возникновения, что помогает в диагностике и устранении сбоев системы.

Таблица 4 – таблица ErrorLog

|  |  |  |  |
| --- | --- | --- | --- |
| Столбец | Тип данных | Ограничения | Описание |
| ErrorID | INT | PRIMARY KEY, AUTO\_INCREMENT | Уникальный идентификатор записи ошибки |
| MessageID | INT | FOREIGN KEY (ссылается на Messages.MessageID) | Идентификатор сообщения, в котором произошла ошибка |
| ErrorDescription | TEXT |  | Описание ошибки |
| ErrorTime | DATETIME | DEFAULT CURRENT\_TIMESTAMP | Время возникновения ошибки, по умолчанию текущее время |

### 4.3 Архитектура программного обеспечения

Архитектура программного обеспечения – это структура программы или вычислительной системы, которая включает программные компоненты, видимые снаружи свойства этих компонентов, а также отношения между ними.

В соответствии с поставленной задачей была разработаны классы, представляющие бизнес-логику приложения.

А также выделены две основные части программы:

* Серверная часть
* Клиентская часть

Схема взаимодействия клиента и сервера представлена на рисунке 4.3.

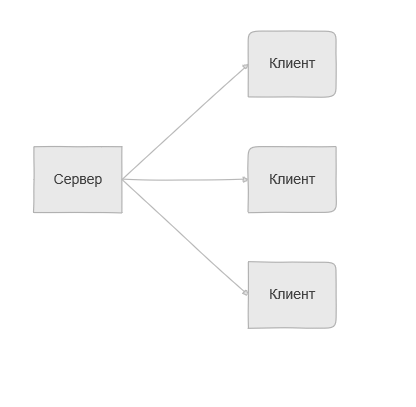


Рисунок 4.3 - Схема взаимодействия клиента и сервера

### 4.4 Проектирование основных классов

## 5 Реализация программного обеспечения

### 5.1 Реализация серверной части

Серверная часть

5.2 Реализация клиентской части

Клиентская часть

### 5.3 Система контроля версий, удаленный репозиторий хранения версий разрабатываемой программы. Стратегия разработки.

Система контроля версий

### 5.4 Безопасность. Шифрование данных. Защита от SQL-инъекций и других уязвимостей.

### 5.5 Механизмы межпроцессного взаимодействия (проецирование файлов, разделяемая память, сокеты, очереди сообщений, сигналы).

### 5.6 Многозадачность и многопоточность, а также средства их синхронизации (семафоры, мьютексы).

### 5.7 Модуль приложения

### 5.8 Пользовательские библиотеки

### 5.9 Жерналирование

### 5.10 Развертывание приложения, создание установочного пакета

## 6 Тестирование программного обеспечения

### 6.1 Тесты для проверки функциональности системы

### 6.2 Тестирование на различных конфигурациях и сценариях использования.

## 7 Документация

### 7.1 Документация по установке и использованию приложения.

### 7.2 Описание структуры кода и основные функции

## **ЗАКЛЮЧЕНИЕ**

Заключение.

## **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. А
2. П
3. Р
4. п

## ПРИЛОЖЕНИЕ А

Графическая часть

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | |
|  |  |  |  |  | КП—1070132211–2025–01 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Махонина |  |  | Диаграмма вариантов использования | Лит | | | Лист | Листов |
| Руковод. | | Азаревич |  |  |  | Д |  | **1** | **5** |
| Консульт. | | Азаревич |  |  | 1 40 01 01 БНТУ  г.Минск | | | | |
| Н.контр. | | Азаревич |  |  |
| Зав.каф. | |  |  |  |

Графическая часть

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | |
|  |  |  |  |  | КП—1070132211–2025–02 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Махонина |  |  | ?????? | Лит | | | Лист | Листов |
| Руковод. | | Азаревич |  |  |  | Д |  | **2** | **5** |
| Консульт. | | Азаревич |  |  | 1 40 01 01 БНТУ  г.Минск | | | | |
| Н.контр. | | Азаревич |  |  |
| Зав.каф. | |  |  |  |

Графическая часть

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | |
|  |  |  |  |  | КП—1070132211–2025–03 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Махонина |  |  | ??????? | Лит | | | Лист | Листов |
| Руковод. | | Азаревич |  |  |  | Д |  | **3** | **5** |
| Консульт. | | Азаревич |  |  | 1 40 01 01 БНТУ  г.Минск | | | | |
| Н.контр. | | Азаревич |  |  |
| Зав.каф. | |  |  |  |

Графическая часть

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | |
|  |  |  |  |  | КП—1070132211–2025–04 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Махонина |  |  | ????????? | Лит | | | Лист | Листов |
| Руковод. | | Азаревич |  |  |  | Д |  | **4** | **5** |
| Консульт. | | Азаревич |  |  | 1 40 01 01 БНТУ  г.Минск | | | | |
| Н.контр. | | Азаревич |  |  |
| Зав.каф. | |  |  |  |

Графическая часть

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | |
|  |  |  |  |  | КП—1070132211–2025–05 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Махонина |  |  | ?????????? | Лит | | | Лист | Листов |
| Руковод. | | Азаревич |  |  |  | Д |  | **5** | **5** |
| Консульт. | | Азаревич |  |  | 1 40 01 01 БНТУ  г.Минск | | | | |
| Н.контр. | | Азаревич |  |  |
| Зав.каф. | |  |  |  |