|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Министерство науки и высшего образования Российской Федерации  Федеральное государственное бюджетное образовательное учреждение высшего образования  «Московский государственный технический университет имени Н.Э. Баумана»  (национальный исследовательский университет)»  Московский техникум космического приборостроения  УТВЕРЖДАЮ  Заместитель директора по УР  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.Н. Ковзель  (подпись, дата)   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**   |  | | --- | | **СЕРВИС АНАЛИЗА И ОБРАБОТКИ ГОЛОСОВЫХ СООБЩЕНИЙ В ДИСКОРД** | |  |   Пояснительная записка  Группа ТИП-82  Листов \_\_ | | | | | Председатель предметной (цикловой) комиссии |  |  | Е.А. Митрошенкова | | (подпись, дата) |  | (ФИО) | | Руководитель разработки  от техникума |  |  | Л.Б. Петренко | | (подпись, дата) |  | (ФИО) | | Рецензент |  |  | Р.В. Мещеряков | | (подпись, дата) |  | (ФИО) | | Руководитель разработки  от предприятия |  |  | Р.Р. Галин | | (подпись, дата) |  | (ФИО) | | Консультант по  экономической части |  |  | С.А. Ковалев | | (подпись, дата) |  | (ФИО) | | Разработчик |  |  | В.Д. Ролдугин | | (подпись, дата) |  | (ФИО) |   Москва, 2022 |

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана»

(национальный исследовательский университет)»

Московский техникум космического приборостроения

Срок окончания проекта «\_\_» июня 20\_\_ г.

Зав. отделением\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О. В. Коротченко Дата «\_\_\_» \_\_\_\_\_\_\_\_\_20\_\_ г.

(подпись)

ЗАДАНИЕ

на выполнение дипломного проекта

Студенту \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ролдугину В.Д \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество)

Тема работы Сервис анализа и обработки голосовых сообщений в дискорд\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Введение

1. Теоретическая обоснование разрабатываемого программного продукта
   1. Описание предметной области
   2. Сравнительный анализ программ-аналогов
   3. Моделирование проектируемой системы
   4. Разработка функциональных требовании к программной системе
   5. Обоснование выбора средств реализации программной системы
   6. Выводы по разделу
2. Разработка программного продукта
   1. Разработка архитектуры программной системы
   2. Разработка структуры данных
   3. Конструирование пользовательского интерфейса
   4. Схемы алгоритма программы и подпрограмм
   5. Отладка и тестирование программы
   6. Руководство пользователя
   7. Вывод по разделу
3. Охрана труда
4. Технико-экономическая часть

Приложения

Презентация

Дата выдачи задания «\_\_\_\_»\_\_\_\_\_\_\_ 20\_\_\_\_\_ г.

Руководитель дипломного проекта от техникума\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата) (ФИО)

Руководитель работы от предприятия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата) (ФИО)

Срок сдачи обучающимся готовой работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ г.

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**СОДЕРЖАНИЕ**

[Введение 6](#_Toc105421891)

[1 Теоретическое обоснование разрабатываемого программного продукта 8](#_Toc105421892)

[1.1 Описание предметной области 8](#_Toc105421893)

[1.2 Сравнительный анализ программ–аналогов 10](#_Toc105421894)

[1.3 Моделирование проектируемой системы 11](#_Toc105421895)

[1.4 Разработка функциональных требований к программной система 14](#_Toc105421896)

[1.4.1 Назначение программы 14](#_Toc105421897)

[1.4.2 Требования к функциональным характеристикам 14](#_Toc105421898)

[1.4.3 Требования к надежности 15](#_Toc105421899)

[1.4.4 Требования к составу и параметрам технических средств 15](#_Toc105421900)

[1.4.5 Требования к информационной и программной совместимости 16](#_Toc105421901)

[1.4.6 Требования к программной документации 16](#_Toc105421902)

[1.5 Обоснование выбора средств реализации программной системы 17](#_Toc105421903)

[1.6 Вывод по разделу 18](#_Toc105421904)

[2 Разработка программного продукта 19](#_Toc105421905)

[2.1 Разработка архитектуры программной системы 19](#_Toc105421906)

[2.2 Разработка структуры данных 22](#_Toc105421907)

[2.3 Конструирование пользовательского интерфейса 25](#_Toc105421908)

[2.4 Схемы алгоритма программы и подпрограмм 2](#_Toc105421909)9

[2.4.1 Схема алгоритма подключения к БД 29](#_Toc105421910)

[2.4.2 Схемы алгоритма команд приложения 30](#_Toc105421911)

[2.4.2.1 Схема алгоритма команды auth 30](#_Toc105421912)

[2.4.2.2 Схема алгоритма команды join\_leave 31](#_Toc105421913)

[2.4.2.3 Схема алгоритма команды all\_us 32](#_Toc105421914)

[2.4.2.4 Схема алгоритма команды give\_elder 33](#_Toc105421915)

[2.4.2.5 Схема алгоритма команды students 34](#_Toc105421916)

[2.4.2.6 Схема алгоритма команды reg 35](#_Toc105421917)

[2.4.3 Схемы алгоритма функций приложения 36](#_Toc105421929)

[2.4.3.1 Схема алгоритма функции message 36](#_Toc105421930)

[2.4.3.2 Схема алгоритма функции connect 37](#_Toc105421931)

[2.4.3.3 Схема алгоритма функции Select\_dict 38](#_Toc105421932)

[2.4.3.4 Схема алгоритма функции speak\_impl 39](#_Toc105421933)

[2.4.3.5 Схема алгоритма функции loadConfig 40](#_Toc105421934)

[2.4.3.6 Схема алгоритма функции leave 41](#_Toc105421935)

[2.4.3.7 Схема алгоритма фунции process\_commands 42](#_Toc105421936)

[2.4.3.8 Схема алгоритма функции transcribe 43](#_Toc105421937)

[2.5 Отладка и тестирование программы 44](#_Toc105421950)

[2.6 Руководство пользователя 50](#_Toc105421951)

[2.6.1 Серверная часть 50](#_Toc105421952)

[2.6.1.1 Проверка и включение службы базы данных 50](#_Toc105421953)

[2.6.1.2 Создание базы данных 52](#_Toc105421954)

[2.6.1.3 Настройка сервера 56](#_Toc105421955)

[2.6.1.4 Запуск и работа с сервером 59](#_Toc105421956)

[2.6.2 Клиентская часть 60](#_Toc105421957)

[2.6.2.1 Настройка клиента 60](#_Toc105421958)

[2.6.3 Команда !auth 64](#_Toc105421957)

[2.6.4 Команда !reg 64](#_Toc105421957)

[2.6.5 Команда !give\_elder 65](#_Toc105421957)

[2.6.6 Команда !join 65](#_Toc105421957)

[2.6.7 Команда !leave 67](#_Toc105421957)

[2.6.8 Команда !all\_us 68](#_Toc105421957)

[2.6.9 Команда !students 69](#_Toc105421957)

[2.7 Вывод по разделу 70](#_Toc105421965)

[3 Охрана труда 71](#_Toc105421973)

[3.1 Техника безопасности при работе на персональном компьютере 71](#_Toc105421974)

[3.2 Требования к помещению (офиса, рабочего места) 73](#_Toc105421975)

[3.3 Мероприятия по противопожарной технике 74](#_Toc105421976)

[4 Технико–экономическая часть 76](#_Toc105421966)

[4.1 Технико – экономические расчеты экономической целесообразности разработки программы 76](#_Toc105421967)

[4.2 Расчетная часть 79](#_Toc105421968)

[4.2.1 Расчет трудоемкости разработанной программы 79](#_Toc105421969)

[4.2.2 Расчет себестоимости разработанной программы 83](#_Toc105421970)

[4.2.3 Анализ возможных путей снижения себестоимости 91](#_Toc105421971)

[4.3 Графическая часть 91](#_Toc105421972)

[Заключение 94](#_Toc105421977)

[Список использованных источников 95](#_Toc105421978)

[Приложение А Листинги программы 97](#_Toc105421979)

[А.1 Листинг команд приложения 98](#_Toc105421980)

[А.1.1 Листинг команды Reg 98](#_Toc105421981)

[А.1.2 Листинг команды Students 103](#_Toc105421981)

[А.1.3 Листинг команды Join\_leave 105](#_Toc105421981)

[А.1.4 Листинг команды Auth 107](#_Toc105421981)

[А.1.5 Листинг команды Give\_Elder 109](#_Toc105421981)

[А.1.6 Листинг команды All\_us 113](#_Toc105421981)

[А.2 Листинги обработки событий в приложении 114](#_Toc105421982)

[А.2.1 Событие Message 114](#_Toc105421983)

[А.2.2 Событие Index 116](#_Toc105421992)

[А.2.3 Событие Ready 117](#_Toc105421992)

[А.2.4 Событие VoiceStateUpdate 118](#_Toc105421992)

[А.2.5 Событие Leave 119](#_Toc105421992)

[А.3 Главный модуль программы Index 120](#_Toc105421995)

[А.4 Листинг класса ConDb 128](#_Toc105421995)

[Приложение Б Результаты выполнения программы 129](#_Toc105422045)

ВВЕДЕНИЕ

В настоящее время видеоконференцсвязь (ВКС) становится основным инструментом современных коммуникаций, а различные встречи довольно часто проходят в формате онлайн. В образовании ВКС также становится все более популярной и востребованной. Система позволяет слушать лекции и заниматься с преподавателями со всего мира в режиме видеосвязи в удобном формате. Устраивать обсуждения, коллоквиумы, семинары в онлайн-формате, где может высказаться любой участник, а также услышать точку зрения остальных посетителей. Становится очень удобным, когда можно просто прочитать всю запись, в текстовом виде, которая была записана в голосовом канале, а не просматривать запись в видео формате целиком и искать важные детали. Удобством является также то, что можно найти информацию при помощи поиска.

Успешное функционирование ВКС невозможно без развитой информационной системы, которая позволяет автоматизировать сбор и обработку данных.

Возникла необходимость разработать данный проект в связи с тем, что ключевые преимущества физических пространств можно перенести в онлайн-среду, а также найти новые инструменты для обеспечения развития и поддержания новых связей. Для перевода мероприятий в дистанционный формат используется специальная платформа и программа работы с ней.

Актуальность работы состоит в том, чтобы уменьшить затраты на просмотр информации по завершении конференции. Это реализуется с помощью специальных библиотек по обработке голоса в текстовый формат, и работой с платформой Dicord.

В качестве языка программирования для реализации данного дипломного проекта был выбран язык программирования JS с применением библиотеки NODE.

В качестве языка системе обработки голоса действительно есть спрос для центров научно-технологического развития или обычных компаний. Система позволяет повысить качество проводимых курсов, мероприятий и проектов.

ООО "ЦИИР" является центром научно-технологического развития, занимающийся образовательной деятельностью, организацией мероприятий и автоматизацией бизнес-процессов, а именно проводит:

* образовательные курсы;
* проекты;
* организация мероприятий;
* автоматизация предприятий и бизнес процессов.

Таким образом, программа, разрабатываемая в дипломном проекте, будет направлена на обеспечение удобного доступа к информации о результатах проведенных конференций. Для этого необходимо решить следующие задачи:

1. изучить библиотеки для работы с дискордом;
2. изучение NODE js;
3. исследовать процессы предметной области (ПрО);
4. описать и спроектировать базу данных (БД);
5. спроектировать архитектуру сервера приложений и клиента;
6. анализ существующих способов распознавания речи;
7. проведение оценки эффективности разработанного метода;
8. разработать и протестировать информационную систему.

1 Теоретическое обоснование разрабатываемого программного продукта

1.1 Описание предметной области

В наше время большинство конференций проходит в онлайн формате, и корректная обработка краткого содержания становиться очень актуальна. В дальнейшем эту информацию можно передать всем, кто не смог быть на конференции.

Discrod – это платформа Voip, – мгновенных сообщений и цифровой дистрибуции, предназначенной для создание сообществ. Пользователи общаются с помощью голосовых вызовов, видео звонков, текстовых сообщений, медиа и файлов в частных чатах или в рамках сообществ, называемых "серверами". Серверы-это набор постоянных чатов и каналов голосового чата. Discrod – имеет ряд преимуществ:

− качество связи – Одновременно в видео звонках Discord может участвовать до 25 человек при хорошем качестве звука;

− быстрота – скорость доставки сообщений выше, чем у аналогов;

− распределение – сервера Discord расположены по всему миру, что повышает отказоустойчивость;

– встроенная функция шумоподавления;

– разделение по темам и ролям;

− отсутствие подписок и рекламы.

Шифраторы речи представляют отдельные устройства или обычные программы для персональных компьютеров или телефонов и позволяют изменять голос налету (для телефонов) или обрабатывать записанную речь как обычный аудиофайл на компьютере. Они не слишком сложны в изготовлении, и потому не сильно дороги. Но при обработке голоса в текстовый формат возникает больше затрат, чтобы обработать его корректно и четко. В связи с чем увеличиваться затраты

При рассмотрении классификации систем распознавания речи следует отметить, что классификация может осуществляться по различным параметрам. По литературным источникам системы распознавания речи можно классифицировать следующим образом:

– в зависимости от размера словаря: системы распознавания речи с ограниченным набором слов; системы со словарем большого размера;

– в зависимости от привязки к диктору: системы, являющиеся дикторозависимыми и дикторонезависимыми;

– в зависимости от типа распознаваемой речи: системы, работающие со слитной речью или раздельной речью.

1.2 Сравнительный анализ программ–аналогов

В результате проведенного анализа среди схожих по функционалу программ была найдена одна система «DiscrodEarsBot».

«DiscordEarsBot» – данный сервис был разработан программистом для обработки голосовых сообщений в Discrod и выводом в чат.

Цель сервиса состоит в том, чтобы сделать разговор видимым для всех участников, которые находиться на данном сервере, но при этом не находиться в голосовом канале, а значит просто могут прочитать весь разговор без звука.

Для удобства пользователей бот имеет команду «/info» которая выводит список возможных команд:

– join–войти в канал и начать запись;

– leave – выйти из канала и прекратить запись.

Из плюсов можно отметить:

− доступный интерфейс;

− полная сводка о сказанных сообщениях.

Минусами данного чат-бота являются:

− неудобный вывод сообщений разговора;

− ошибки при вылете из канала.

1.3 Моделирование проектируемой системы

Первый шаг в анализе функций системы представлен при помощи диаграмм IDEF0 –это построение контекстной диаграммы. Контекстная диаграмма в самом общем виде описывает деятельность ИС. Единственный функциональный блок, которой присутствует на контекстной диаграмме, отражает основную производственную деятельность. Данный блок показывает, что система получает на входе голосовое сообщение и обрабатывает его в текстовый формат.

На основании анализа предметной области были сформулированы диаграммы прецедентов для каждой роли будущей системы.

На рисунке 1.1 представлена диаграмма IDEF0.



Рисунок 1.1 – Контекстная диаграмма IDF0

Теперь основную производственную деятельность нужно разбить на составные функции. Для этого требуется произвести декомпозицию.

Результаты построения диаграммы декомпозиции первого уровня представлены на рисунке 1.2.



Рисунок 1.2 – Диаграмма декомпозиций первого уровня

На рисунке 1.3 представлена контекстная схема DFD.



Рисунок 1.3 – Контекстная диаграмма DFD

На рисунке 1.4 представлена декомпозиция схема DFD, показывающая полный цикл вывода информации.



Рисунок 1.4 – Схема потоков данных разрабатываемого ПО

На рисунке 1.5 представлена функциональная схема Use case показывающая полный цикл вывода информации.



Рисунок 1.5 – Диаграмма Use case

1.4 Разработка функциональных требований к программной система

1.4.1 Назначение программы

Данная ИС используется для учета и обработки голосовых сообщений, а также преобразование записей в текстовый формат, может быть использована в ЦИИР.

1.4.2 Требования к функциональным характеристикам

В рамках управления предприятиями ИС должна поддерживать следующий функционал:

1. начать запись всех участников;
2. при начатой записи обрабатывать голосовые сообщения в текстовый формат в определенном виде;
3. учет история отправки сообщений;
4. завершить запись.
5. аутентификацию пользователей – авторизации пользователей;
6. авторизацию пользователей – разграничение прав пользователей на выполнение действий в системе на уровне сервера;
7. выдача различных ролей в системе при необходимом уровне доступа;
8. просмотр всех пользователей;
9. просмотр студентов – в вашей группе или определённых группах.

Данные требования описывают весь функционал, связанный с работой с пользователями, а также с аутентификацией и авторизацией пользователей.

1.4.3 Требования к надежности

Надежность технических средств обеспечивается использованием сертифицированных средств вычислительной техники и их комплектующих. Надежность ПО обеспечивается использованием сертифицированных ОС, программных средств, используемых при разработке программного продукта. Для обеспечения надежности прикладного ПО, необходимо следовать определенным требованиям:

1. предусмотреть обработку исключений, связанных с вводом пользователем неправильных данных, обращением программы к несуществующим файлам;
2. выполнить обязательное освобождение памяти после использованных и впоследствии ненужных динамически выделенных данных;
3. прикладные программы не должны модифицировать свой код или коды других программ во время работы.

1.4.4 Требования к составу и параметрам технических средств

Для корректной работы серверной части необходимо учесть следующие минимальные системные и программные требования:

1. операционная система семейства Windows, начиная с 7 версии;
2. оперативная память от 512 MB;
3. свободное место на диске от 700 MB;
4. сервер БД MySQL версии 8.0.27.

Соблюдение данных требований необходимо для работы системы.

1.4.5 Требования к информационной и программной совместимости

Система должна работать под управлением 64-разрядной операционной системы Windows. Для корректной работы программы на компьютер должна быть установлена СУБД MySQL версии 8.0.27.

1.4.6 Требования к программной документации

Программа должны быть интуитивно понятной и расширяемой при необходимости. Разработанные программные модули (программа) должны быть самодокументированы, т.е. листинг программы должен содержать все необходимые комментарии.

В состав сопровождающей документации должны входить:

1. пояснительная записка, содержащая описание разработки:
   * схемы алгоритма программы;
   * отладка программы;
   * тестирование программы;
   * руководство пользователя;
2. приложения:
   * листинг программы;
   * результаты выполнения программы.

Разрабатываемая программа должна включать подсказки пользователю для более удобной работы.

1.5 Обоснование выбора средств реализации программной системы

Для разработки был выбран язык программирования js, среда выполнения Node, в качестве СУБД будет использован MySQL.

[JavaScript](https://developer.mozilla.org/ru/docs/Glossary/JavaScript)  — это полноценный [динамический язык программирования](https://developer.mozilla.org/ru/docs/Glossary/Dynamic_programming_language), который применяется к [HTML](https://developer.mozilla.org/ru/docs/Glossary/HTML) документу, и может обеспечить динамическую интерактивность на веб-сайтах. Его разработал Brendan Eich, сооснователь проекта Mozilla, Mozilla Foundation и Mozilla Corporation.

Node.js — это система, которая исполняет JavaScript отдельно от вашего браузера. Можно сказать, что это самостоятельная среда для выполнения JavaScript.

Node.js можно установить на сервер (так же, как Python) и исполнять на нём ваш код, отдавая результат исполнения пользователям. На нём же можно делать отдельные приложения, используя дополнительные фреймворки.

SQL — это язык структурированных запросов, который позволяет взаимодействовать между вами и базой данных. Есть множество других СУРБД, которые также используют язык SQL. В том числе для названия, например, PostgreSQL.

В целом SQL — один самых популярных языков, использующихся в базах данных. Так как с его помощью можно все манипуляции с данными проходят эффективно. Это важно для скорости работы и стабильности базы.

Описанные средства достаточны и содержат весь необходимый функционал для разработки дипломного проекта.

1.6 Вывод по разделу

В данном разделе был проведен анализ предметной области дипломной работы, были описаны более подробно объект и предмет исследования, были спроектированы основные диаграммы прецедентов, выделены функциональные требования, проведен анализ аналогичного программного обеспечения и описаны средства, что будут использованы в дальнейшем для разработки. На основании собранной информации можно начать проектирование и разработку ИС.

2 Разработка программного продукта

2.1 Разработка архитектуры программной системы

В результате анализа предметной области была спроектирована следующая Архитектура ИС, представленная на рисунке 2.1.



Рисунок 2.1 – Архитектура ИС

ИС имеет трёхзвенную архитектуру, состоящую из сервера БД, сервера приложений и клиента. На схеме представлены модули, входящие в клиент и сервер. Подробное описание данных модулей приведено в таблицах 2.1 и 2.2

Таблица 2.1 – Модули сервера приложений

|  |  |
| --- | --- |
| Название | Описание |
| Domain | Содержит все модели данных, используемые сервером приложений и интерфейсы сервисов, что необходимо реализовать для корректной работы сервера |
| NODE | Реализует методы, к которым обращается клиент |
| Host | Консольное приложение в рамках которого работает NODE |

Таблица 2.2 – Модули клиента

|  |  |
| --- | --- |
| Название | Описание |
| Domain | Содержит все модели данных, используемые клиентом и интерфейсы сервисов, что необходимо реализовать для корректной работы клиента |
| NODEAPI | Содержит реализацию сервисов, описанных в Domain. Служит промежуточным звеном между клиентом и сервером приложений |
| Discord js | Клиент, описывает пользовательский интерфейс платформы discord и клиентскую логику работы с данными, получаемыми от сервера приложений |

Данная архитектура облегчает процесс замены модулей, сводя всю работу к реализации необходимых интерфейсов.

Клиент в свою очередь имеет определенную структуру страниц. Структура страниц клиента представлена на рисунке 2.2.



Рисунок 2.2 – Структура страниц клиента

**2.2 Разработка структуры данных**

В ходе проектирования БД была разработана схема данных, представленная на рисунке 2.3.

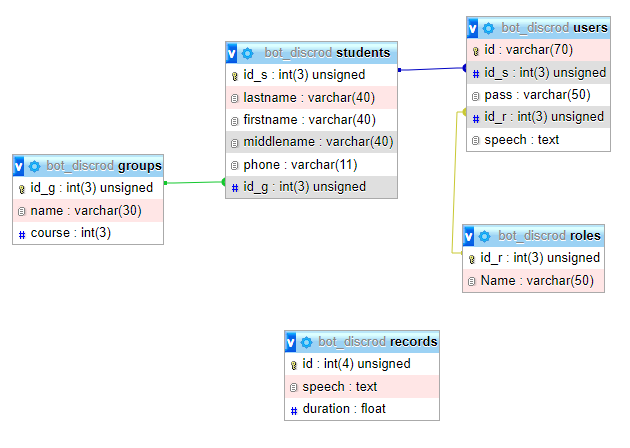


Рисунок 2.3 – Схема данных БД

Подробное описание сущностей БД представлено в таблицах 2.3 – 2.7.

Таблица 2.3 – Сущность «groups»

|  |  |  |
| --- | --- | --- |
| Имя поля | Тип данного | Примечание |
| Id\_g | int(3) | Идентификационный номер группы |
| name | varchar(30) | Название метода |
| course | int(3) | курс |

Таблица 2.4 – Сущность «records»

|  |  |  |
| --- | --- | --- |
| Имя поля | Тип данного | Примечание |
| id | int(4) | Идентификационный номер роли записи |
| speech | text | Все записи проведённые во время конференции |
| duration | float | Время конференции |

Таблица 2.5 – Сущность «roles»

|  |  |  |
| --- | --- | --- |
| Имя поля | Тип данного | Примечание |
| id\_r | int(3) | Идентификационный номер роли |
| Name | varchar(50) | Название роли |

Таблица 2.6 – Сущность «students»

|  |  |  |
| --- | --- | --- |
| Имя поля | Тип данного | Примечание |
| id | int | Идентификационный номер пользователя |
| lastname | varchar(40) | Фамилия пользователя |
| firstname | varchar(40) | Имя пользователя |
| midlename | varchar(40) | Отчество пользователя |
| phone | varchar(11) | Телефон |
| id\_g | int(3) | Идентификационный номер группы |

Таблица 2.7 – Сущность «users»

|  |  |  |
| --- | --- | --- |
| Имя поля | Тип данного | Примечание |
| id | varchar(70) | Идентификационный номер пользователя |
| id\_s | int(3) | Идентификационный номер студента |
| pass | varchar(50) | Пароль |
| id\_r | int(3) | Идентификационный номер роли |
| speech | text | Записи пользователя |

**2.3 Конструирование пользовательского интерфейса**

Клиент реализует готовую команду для входа в систему, что содержит ввод пароля, вид которой представлен на рисунке 2.4.

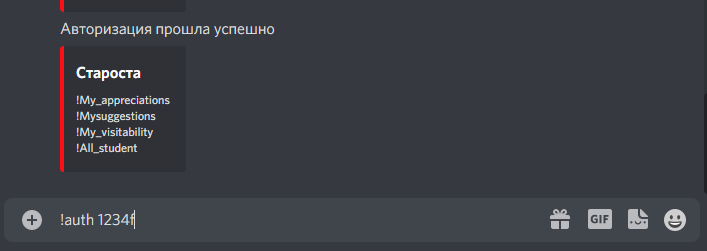


Рисунок 2.4 – Команда входа в систему

После входа в систему пользователь получает доступные команды в зависимости от роли.

Клиент реализует готовую команду для регистрации в системе, что содержит ввод группы, фамилии, имени и пароля, вид представлен на рисунке 2.5.

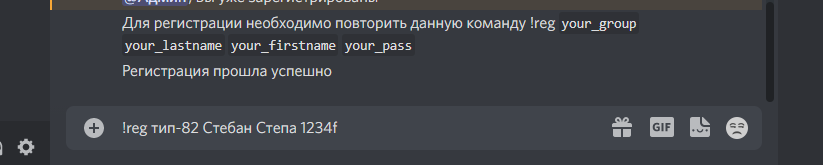


Рисунок 2.5 – Команда регистрации в системе

После регистрации в системе пользователь наблюдает уведомление об успешном завершении.

Клиент реализует готовую команду для записи конференции, вид представлен на рисунке 2.6.

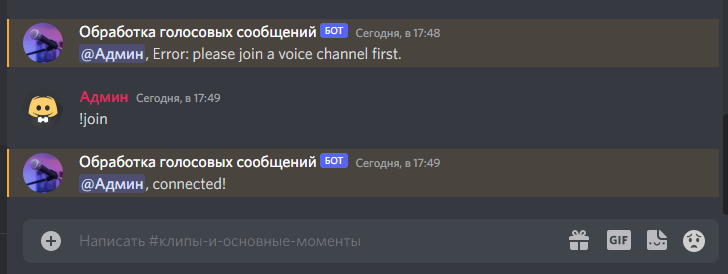


Рисунок 2.6 – Команда записи конференции

После запуска записи, система появляется в голосовом канале, и начинает записывать через разные потоки всех пользователей, находящихся там.

Клиент реализует готовую команду для завершения записи конференции, вид представлен на рисунке 2.7.



Рисунок 2.7 – Команда завершения конференции

При написании команды завершения конференции, когда система не находиться внутри какого-либо канала, то система напишет, что нельзя завершить конференцию, не находясь в голосовом канале. Если произойдет ситуация, когда пригласивший забудет выключить систему, то присутствуют два варианта:

1. если пользователь выйдет из голосового канала, не выполнив команду завершении конференции, то запись автоматически выключится;
2. если пользователь отошёл от рабочего места и не покинул голосовой канал, то в скором времени платформа discord выгонит участника за бездействие, и тем самым сработает алгоритм, который автоматически выключит систему.

Клиент реализует готовую команду для просмотра студентов своей группы, вид представлен на рисунке 2.8.

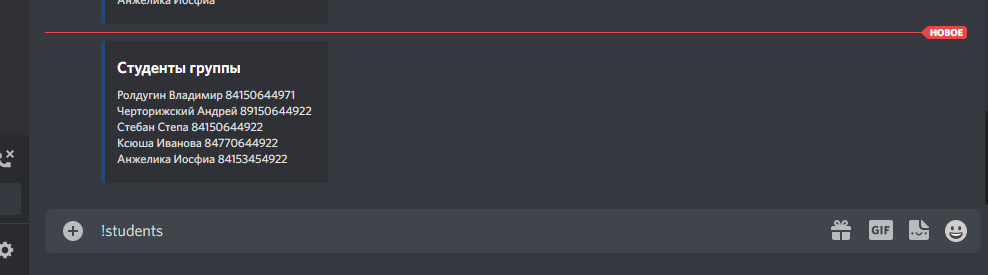


Рисунок 2.8 – Команда просмотра студентов

Клиент реализует готовую команду для просмотра всех пользователей, вид представлен на рисунке 2.9.

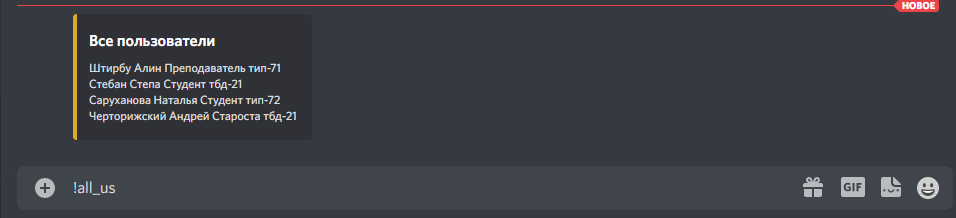


Рисунок 2.9 – Команда просмотра всех пользователей

Клиент реализует готовую команду для выдачи ролей, вид представлен на рисунке 2.10.

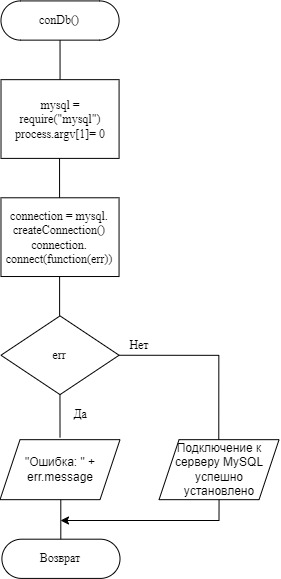


Рисунок 2.10 – Команда выдачи роли

После завершения, придет успешное сообщение с тем, что роль изменилась, при неуспешном, сообщение что только преподаватели могут выдать роль.

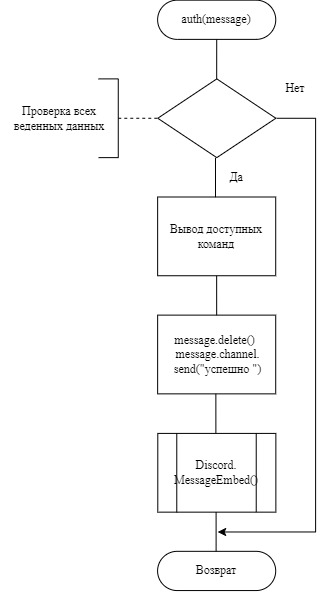
**2.4 Схемы алгоритма программы и подпрограмм**

**2.4.1 Схема алгоритма подключения к БД**

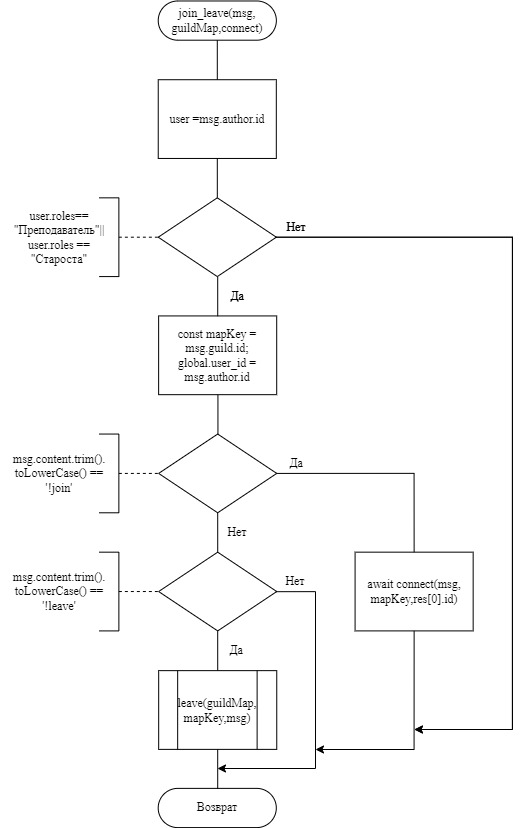


**2.4.2 Схемы алгоритмов команд приложения**

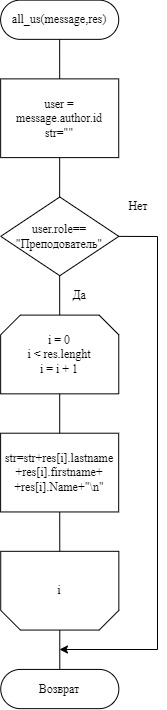
**2.4.2.1 Схема алгоритма команды auth**



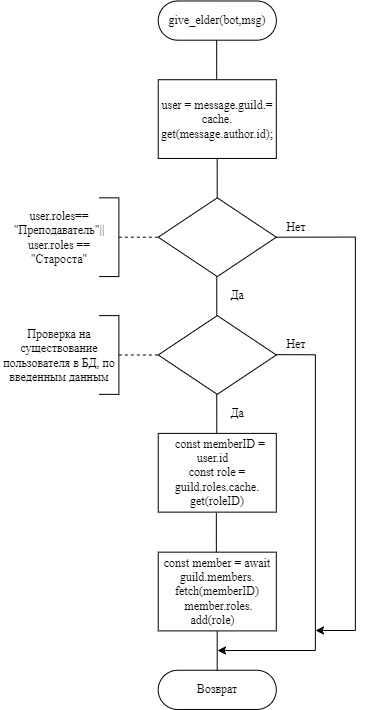
**2.4.2.2 Схема алгоритма команды join\_leave**



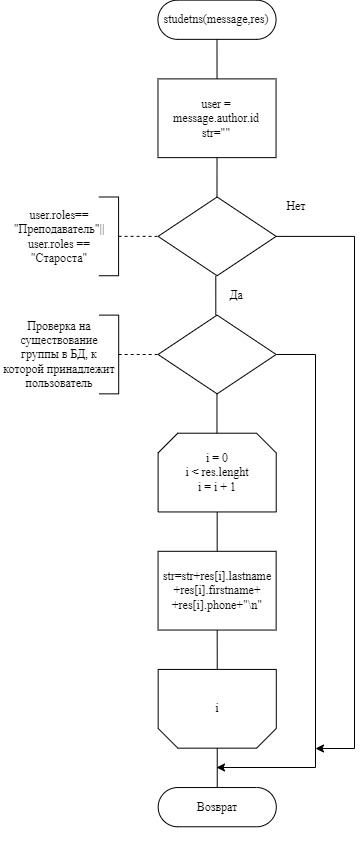
**2.4.2.3 Схема алгоритма команды all\_us**



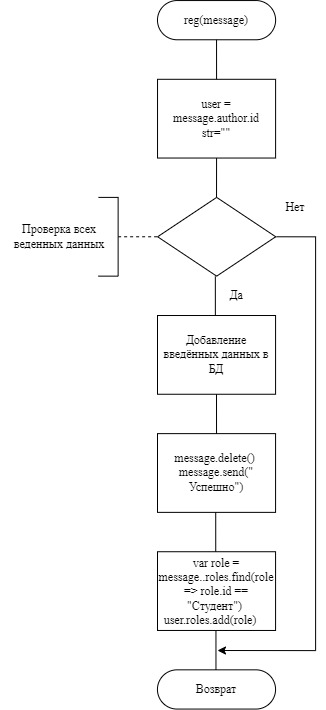
**2.4.2.4 Схема алгоритма команды give\_elder**



**2.4.2.5 Схема алгоритма команды students**

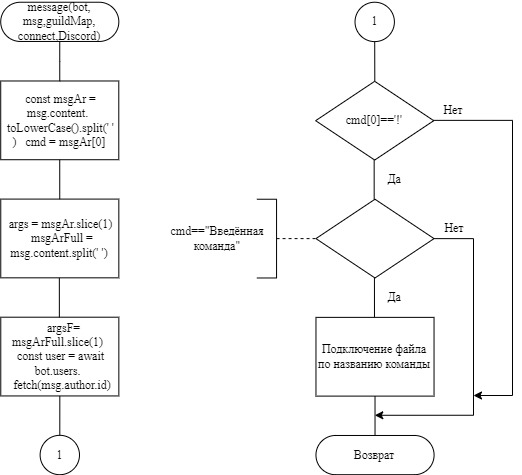


**2.4.2.6 Схема алгоритма команды reg**

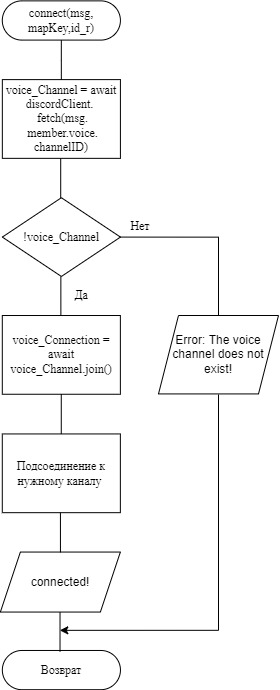


**2.4.3 Схемы алгоритмов функций приложения**

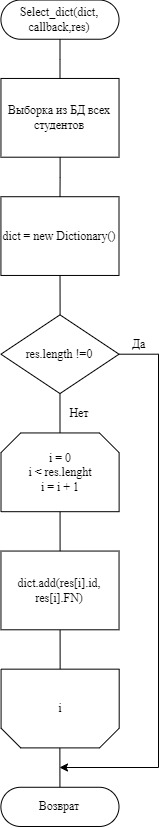
**2.4.3.1 Схема алгоритма функции message**



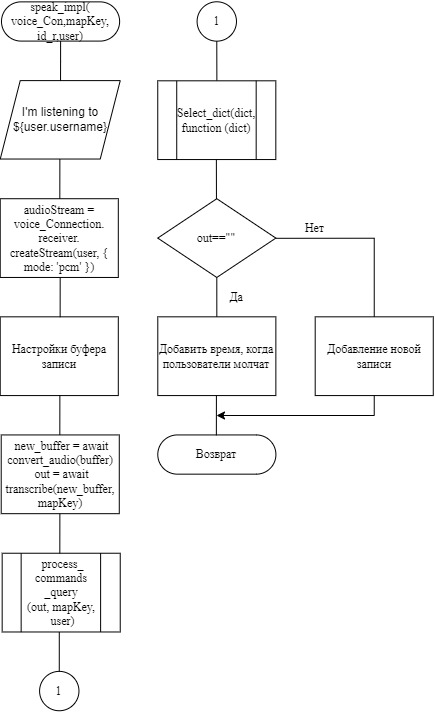
**2.4.3.2 Схема алгоритма функции connect**



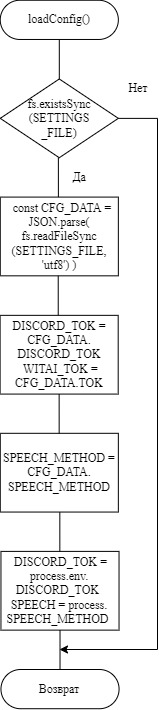
**2.4.3.3 Схема алгоритма функции Select\_dict**



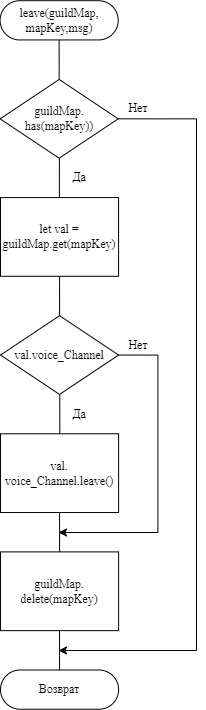
**2.4.3.4 Схема алгоритма функции speak\_impl**



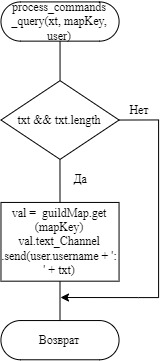
**2.4.3.5 Схема алгоритма функции loadConfig**



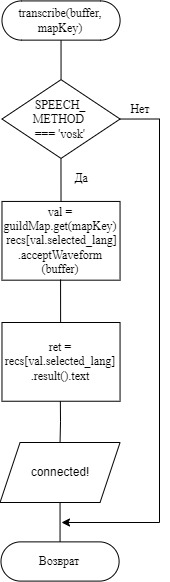
**2.4.3.6 Схема алгоритма функции leave**



**2.4.3.7 Схема алгоритма функции process\_commands**



**2.4.3.8 Схема алгоритма функции transcribe**



2.5 Отладка и тестирование программы

Отладкой называется процесс локализации и устранения ошибок в программном коде проекта. Локализация ошибок – это процесс определения оператора или последовательности действий, выполнение которых вызывает нарушение нормального вычислительного процесса программы. Отладка часто производится с использованием отладчиков. Отладчиком называют средство разработки, присоединяемое к работающему приложению, позволяющее проверить программный код. Visual Studio Code предоставляет свой собственный, встроенный отладчик.

Ошибки, возникающие при отладке, классифицируют на следующие группы:

1. Синтаксические ошибки – ошибки в записи программного кода. При использовании IDE зачастую отлавливаются до компиляции. Также отлавливаются в процессе компиляции/трансляции/интерпретации (в зависимости от языка разработки).

Примеры ошибок, допущенных при разработке:

* ошибки в названиях переменных, полей, свойств, методов;
* ошибки в ключевых словах.

Ошибка:

val.v0ice\_Channe1l leave(

Исправленный вариант:

val.voice\_Channel.leave()

1. Ошибки компоновки – ошибки связей, обнаруживаемые компоновщиком (редактором связей). Как и синтаксические ошибки, при использовании IDE зачастую обнаруживаются до компиляции. К таким ошибкам относят, к примеру несовпадение списка параметров методов (функций, процедур).

Пример ошибки, допущенной при разработке:

* Не добавлены ожидаемые параметры.

Ошибка:

con.query('SELECT \* FROM `users` WHERE `users`.`id` = ? AND `users`.`pass` = ? ', [], async (err,res,fields)=>

Исправленный вариант:

con.query('SELECT \* FROM `users` WHERE `users`.`id` = ? AND `users`.`pass` = ? ', [message.author.id, cont], async (err,res,fields)=>

1. Ошибки выполнения – ошибки, обнаруживаемые в процессе выполнения программы. Данные ошибки труднее отлавливаются ввиду того, что часто не зависят от самой программы, а от сторонних компонентов, к примеру, от аппаратных средств или сторонних сервисов, к примеру, сетевых или интернет сервисов, а также ввиду того, что данные ошибки могут возникнуть вследствие неправильного понимания используемых средств или используемого метода решения задачи.

Пример ошибки, допущенной при разработке:

* обращение к серверу приложений при выключенном сервере БД.

Все ошибки обнаруженные в ходе разработки были исправлены, для удостоверения в корректности работы разработанной ИС было проведено тестирование.

Тестирование – это набор процедур и действий, предназначенных для демонстрации правильности работы программы в заданных режимах и внешних условиях.

Цель тестирования – выявить наличие ошибок или продемонстрировать их отсутствие.

Процесс тестирования был произведен в трёх этапах:

1. проверка поведения программы в нормальных условиях;
2. проверка поведения программы в экстремальных условиях;
3. проверка поведения программы в исключительных ситуациях.

Каждый этап предполагает использование специфических для него наборов данных.

Для успешной работоспособности ИС будут проведены несколько тестовых наборов из проведенного тестирования:

1. Тестовый набор 1.

Описание – добавление пользователя с неправильной группой.

Ожидаемый результат – ошибка «такой группы не существует».

Полученный результат представлен на рисунке 2.11.

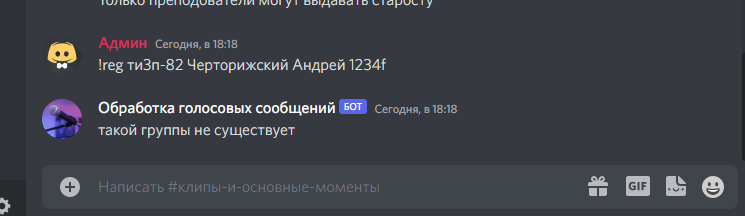


Рисунок 2.11 – Результат тестирования

1. Тестовый набор 2.

Описание – добавление пользователя с неправильным именем в системе.

Ожидаемый результат – ошибка «такого студента не существует».

Полученный результат представлен на рисунке 2.12.

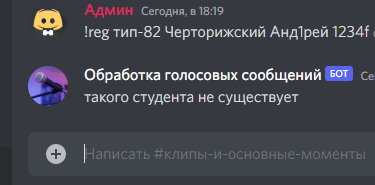


Рисунок 2.12– Результат тестирования

1. Тестовый набор 3.

Описание – добавление пользователя с неправильной фамилией в системе.

Ожидаемый результат – ошибка «такого студента не существует».

Полученный результат представлен на рисунке 2.13.

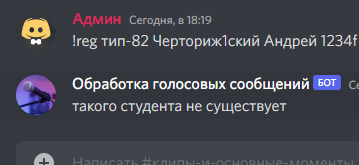


Рисунок 2.13 – Результат тестирования

1. Тестовый набор 4.

Описание – регистрация неправильного формата.

Ожидаемый результат – ошибка «Для регистрации необходимо повторить данную команду !reg your\_group your\_lastname your\_firstname your\_pass».

Полученный результат представлен на рисунке 2.14.

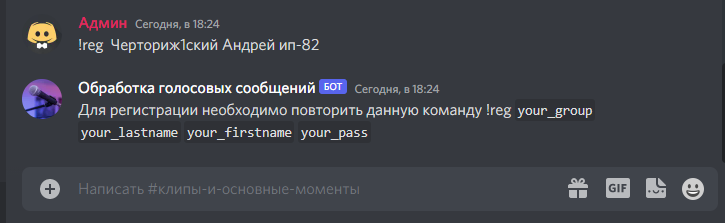


Рисунок 2.14 – Результат тестирования

1. Тестовый набор 5.

Описание – добавление старосты в неправильном формате.

Ожидаемый результат – добавление новой записи.

Полученный результат представлен на рисунке 2.15.

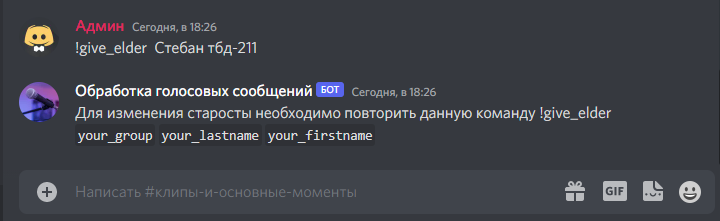


Рисунок 2.15 – Результат тестирования

1. Тестовый набор 6.

Описание – ввод некорректной длины данных.

Ожидаемый результат – ошибка «Недопустимая длина данных».

Полученный результат представлен на рисунке 2.16.

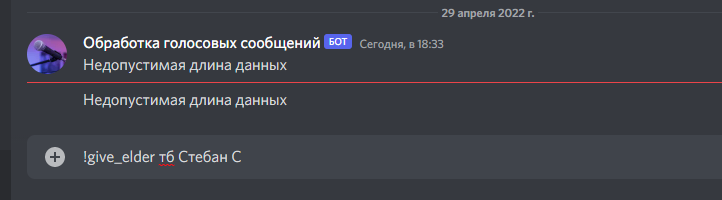


Рисунок 2.16 – Результат тестирования

1. Тестовый набор 7.

Описание – попытка начать запись конференции, не находясь внутри голосового канала.

Ожидаемый результат – ошибка «Error: please join a voice channel first.».

Полученный результат представлен на рисунке 2.17.

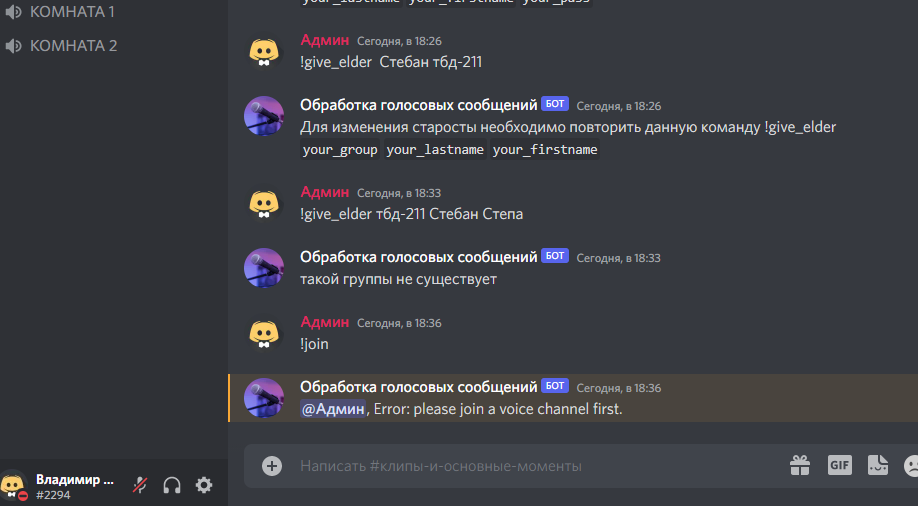


Рисунок 2.17 – Результат тестирования

1. Тестовый набор 8.

Описание – завершение не начатой конференции.

Ожидаемый результат – ошибка «Cannot leave because not connected.».

Полученный результат представлен на рисунке 2.18.

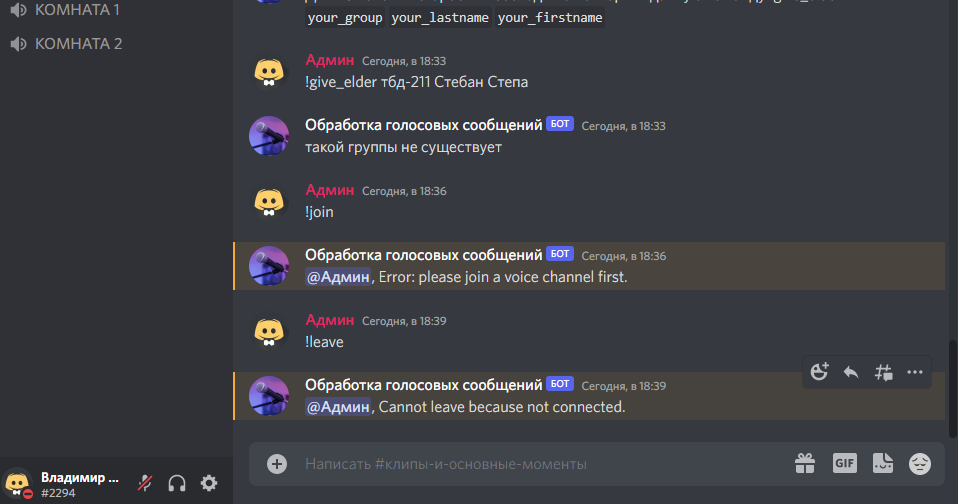


Рисунок 2.18 – Результат тестирования

1. Тестовый набор 9.

Описание – ввод несуществующей команды.

Ожидаемый результат – ошибка «Извините, такой команды не найдено или не существует».

Полученный результат представлен на рисунке 2.19.

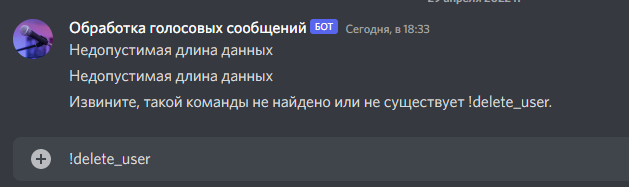


Рисунок 2.19 – Результат тестирования

1. Тестовый набор 10.

Описание – повторная регистрация на такой же id.

Ожидаемый результат – ошибка «Вы уже зарегистрированы».

Полученный результат представлен на рисунке 2.20.

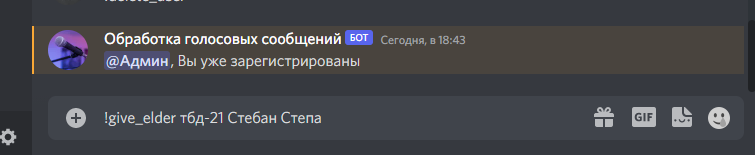


Рисунок 2.20 – Результат тестирования

На основании произведенных тестов и полученных результатов можно сделать вывод, что программа работает корректно.

2.6 Руководство пользователя

Данная ИС обеспечивает обработку голосовых сообщений в текстовый формат и хранение данных о пользователях, конференциях, группах, студентах. Также с помощью ИС можно записывать конференции, которые одновременно обрабатывают всех пользователей, находящихся внутри голосового канала.

2.6.1 Серверная часть

Данный подраздел описывает серверную часть ИС. Для её корректной работы необходимо учесть следующие минимальные системные и программные требования:

1. операционная система семейства Windows, начиная с 7 версии;
2. оперативная память от 512 MB;
3. свободное место на диске от 700 MB;
4. сервер БД MySQL версии 8.0.27;
5. платформа Node.js.

Соблюдение данных требований необходимо для работы сервера.

2.6.1.1 Проверка и включение службы базы данных

Необходимо учитывать, что перед запуском сервера приложений, сервер БД (служба сервера БД) должна быть в включенном состоянии. Проверить это можно открыв окно «Службы», которое можно вызвать путём нажатия комбинации Win + R и вводом в открывшееся поле «services.msc», данный процесс представлен на рисунках 2.21 и 2.22.

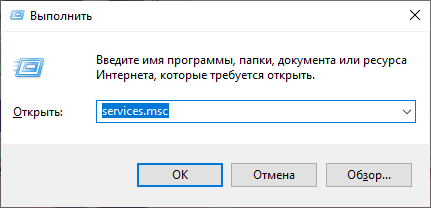


Рисунок 2.21 – Ввод ключевого слова вызова окна «Службы»

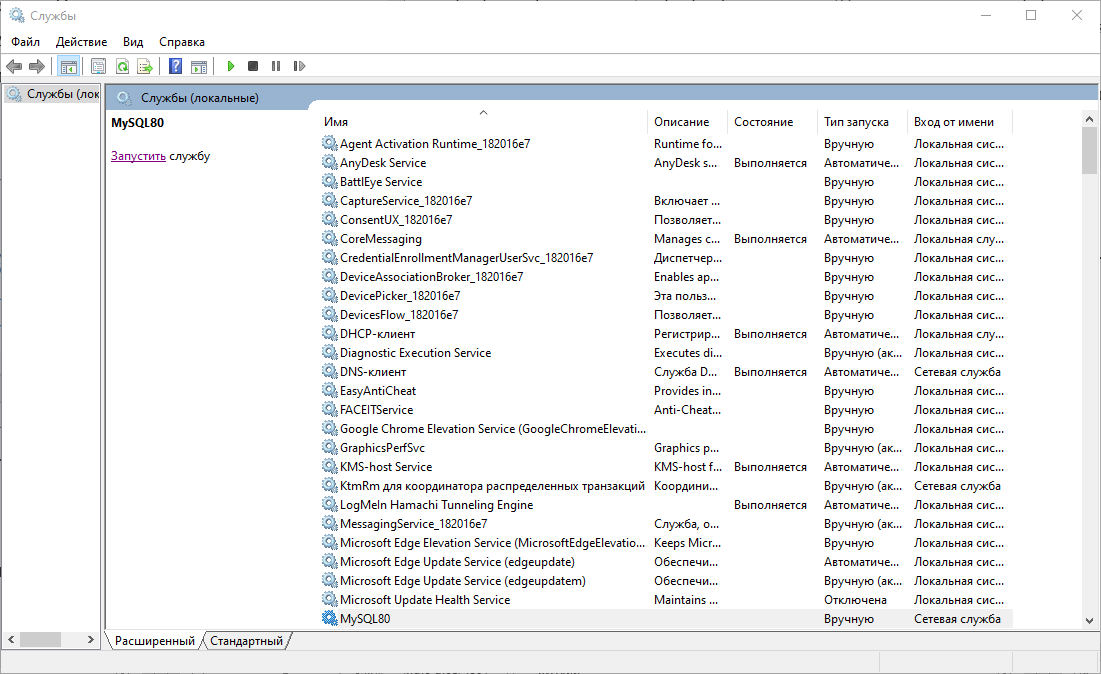


Рисунок 2.22 – Окно «Службы»

В открывшемся окне необходимо найти службу, отвечающую за сервер MySQL, и удостовериться что она включена, в противном случае включить её (рисунок 2.23).

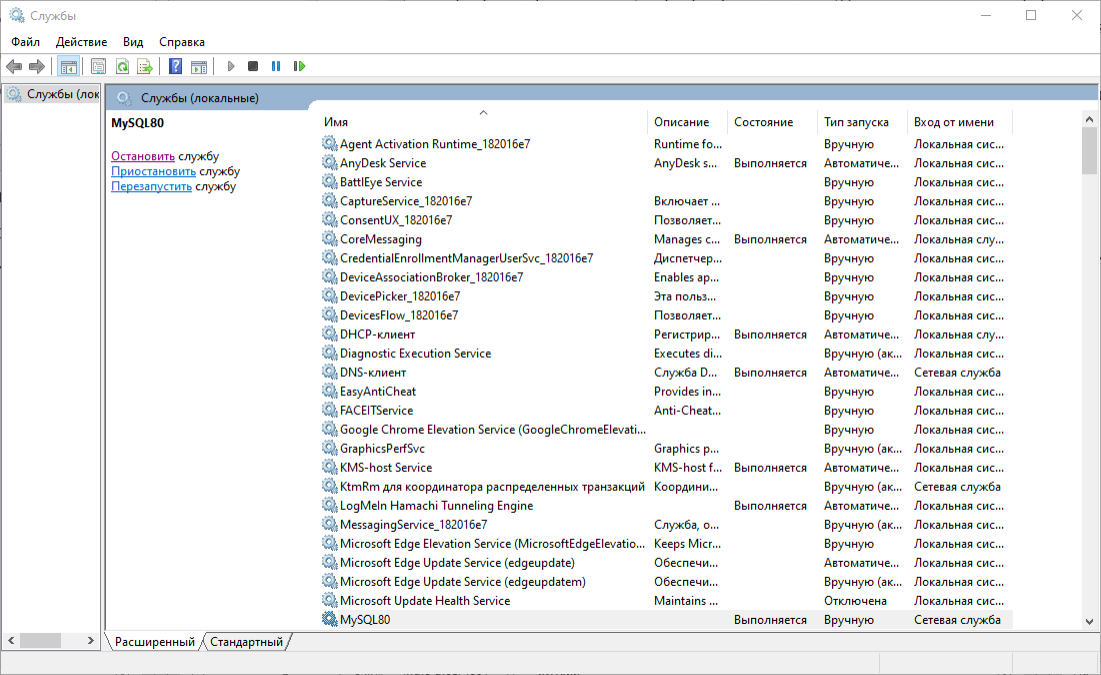


Рисунок 2.23 – Окно со включенной службой MySQL

2.6.1.2 Создание базы данных

Для создания БД необходимо выполнить её скрипт создания (инструкции создания БД), предоставляемый в архиве, в котором распространяется ИС. Скрипт находится в папке «DB», под называнием «bot\_discrod.sql». Изображение с местонахождением скрипта представлено на рисунке 2.24.

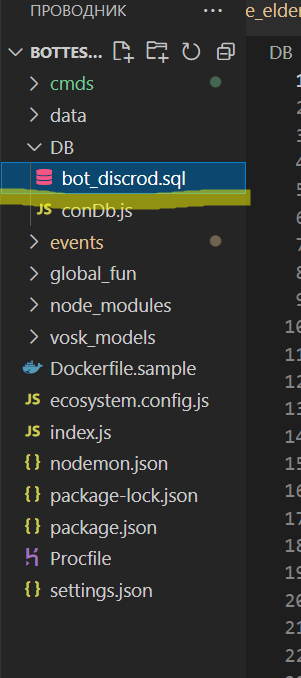


Рисунок 2.24– Местонахождение скрипта создания БД

Далее необходимо открыть MAMP и соединиться с сервером БД. При возникновении ошибок с соединением к БД необходимо посмотреть пункт 2.6.1.1.

После соединения нужно экспортировать файл скрипта в меню, пункт «Import» (рисунок 2.25).

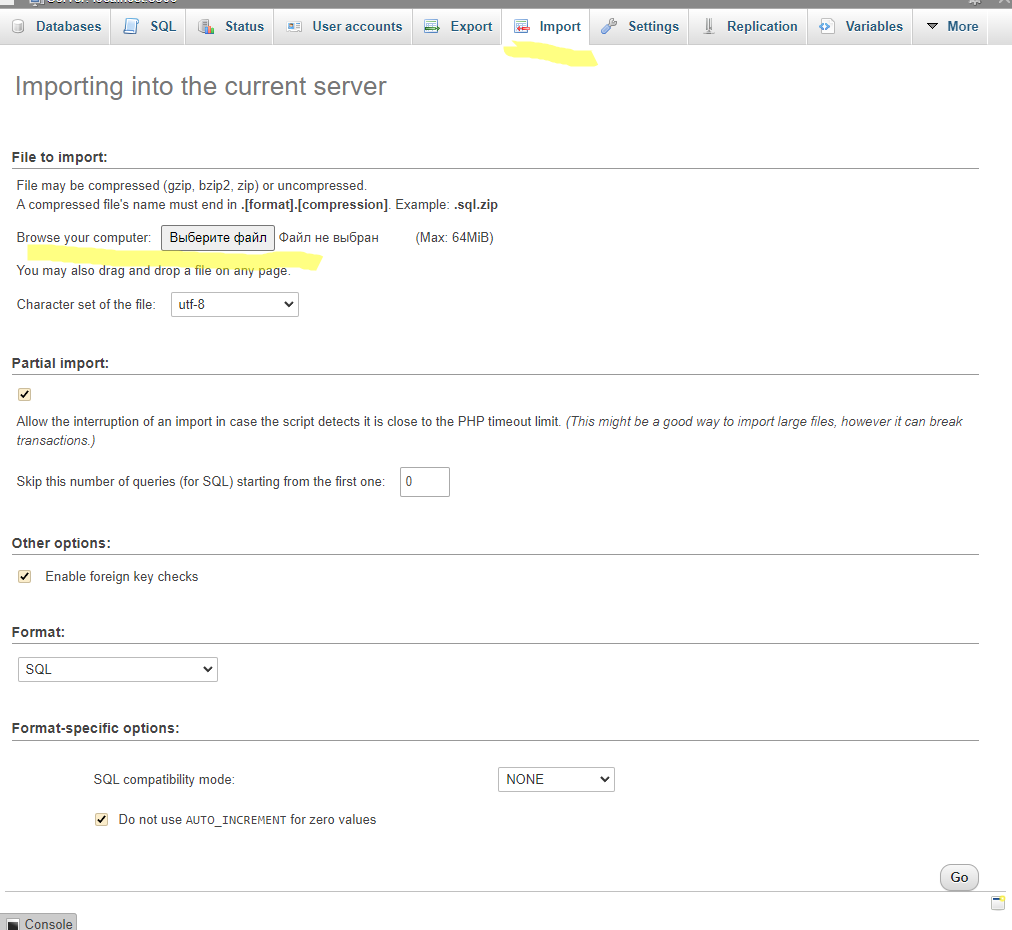


Рисунок 2.25 – Пункт открытия скрипта

После открытия нужно выполнить скрипт нажав на иконку «Go», данное действие представлено на рисунке 2.26.

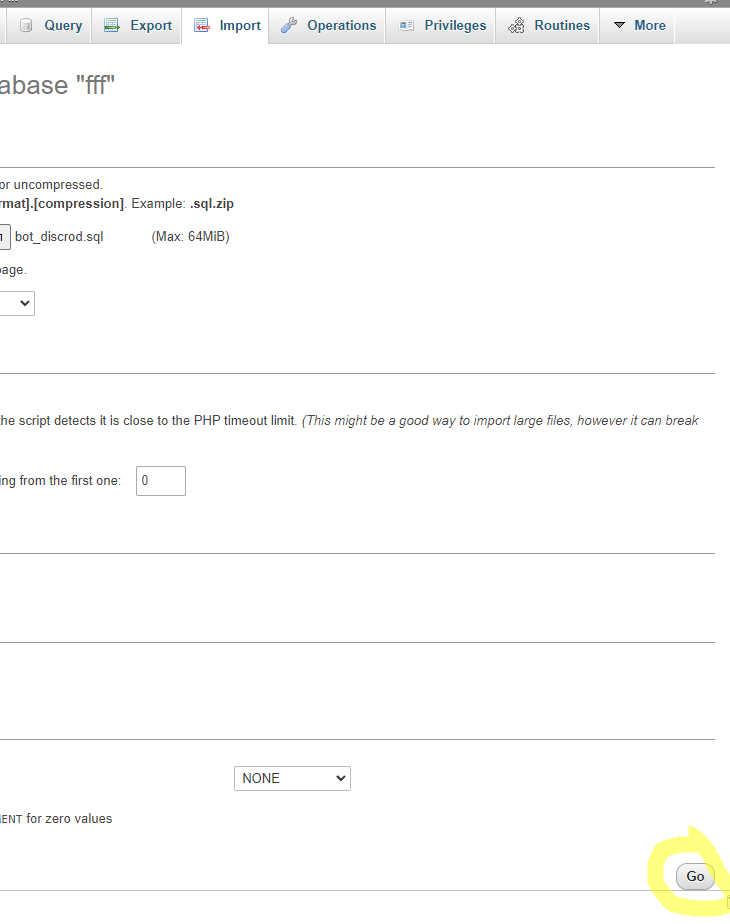


Рисунок 2.26 – Выполнение скрипта создания БД

После выполнения в нижней части окна будут выведены результаты выполнения скрипта (рисунок 2.27).

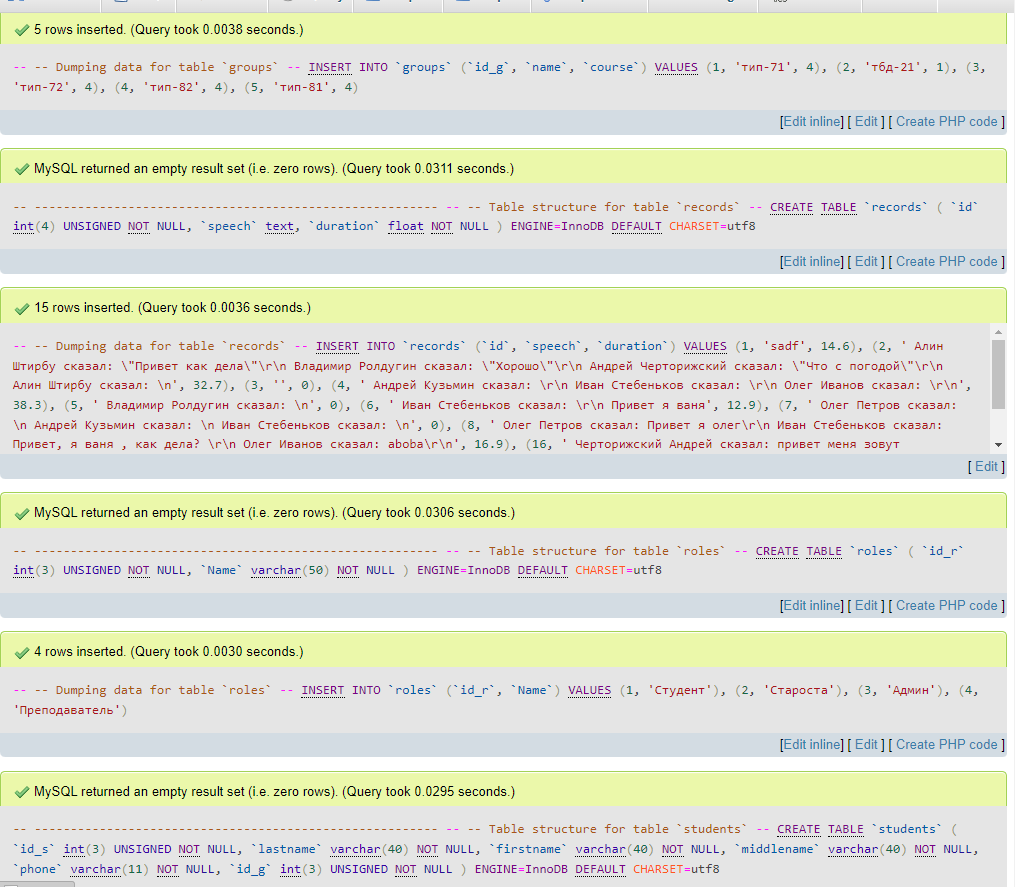


Рисунок 2.27 – Результаты выполнения скрипта

2.6.1.3 Настройка сервера

Для настройки сервера необходимо открыть файл «conDb.js», находящийся в папке «Server», используя текстовый редактор. Данное действие может понадобиться при первом запуске, а также при изменении данных соединения или данных БД.

Следует учитывать, что любые изменения, производимые в конфигурационном файле, стоит производить при закрытой консоли сервера, запущенный сервер внесенные в конфигурационный файл изменения учитывать не будет.

Для изменения данных связи с БД необходимо найти узел, названный «connection» (рисунок 2.28).

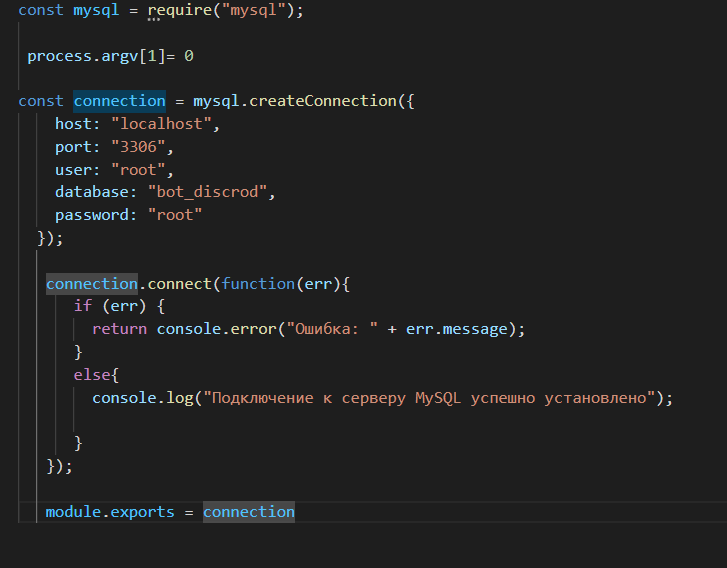


Рисунок 2.28 – Узел «connection»

В данном узле наибольший интерес представляет строка соединения (рисунок 2.29).

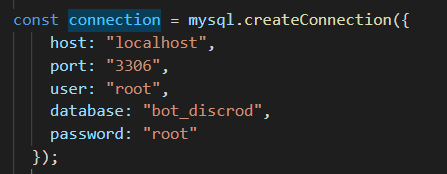


Рисунок 2.29 – Строка соединения с БД

Данная строка содержит необходимые для соединения с БД данные. Изменяя их, можно привести работу сервера в негодность если данные указаны неправильно. Данная строка содержит следующие параметры:

1. server – узел (доменное имя или IP–адрес) на котором развернута БД. Если БД находится на той же машине, на которой будет запущен сервер приложений можно использовать «localhost» или «127.0.0.1» для обозначения локальной машины;
2. user id – пользователь (логин) БД от имени которого будет работать сервер приложений. При использовании ново–созданного пользователя необходимо учесть, что сервер приложений требует права на чтение, добавление, изменение и удаление (DML – инструкции) данных из всех таблиц БД;
3. password – пароль указанного в user id пользователя;
4. database – название БД, которая будет использоваться для работы ИС.

Адрес машины в локальной сети можно найти, использовав команду «ipconfig» в командной строке. Командную строку можно открыть, нажав сочетание клавиш Win + R и вводив «cmd» в окно выполнения, данный процесс представлен на рисунках 2.30 и 2.31.

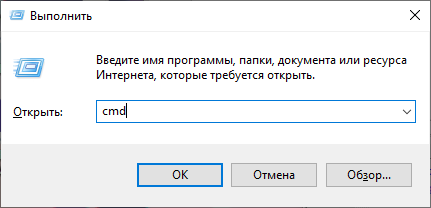


Рисунок 2.30 – Вызов командной строки

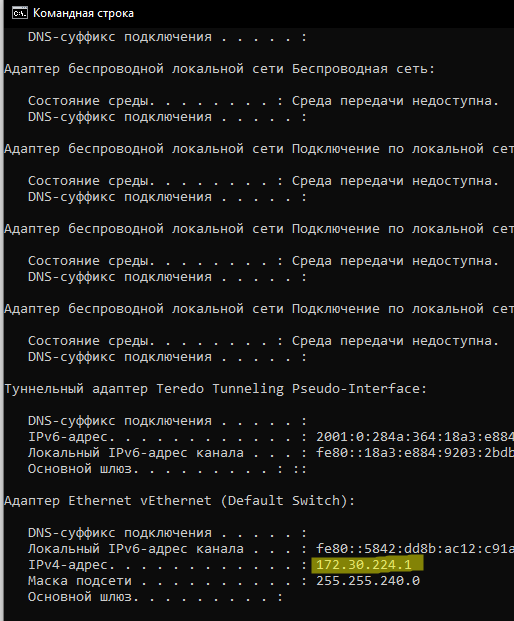


Рисунок 2.31 – Вызов команды «ipconfig» и нахождение адреса машины

В ходе настройки соединения также не стоит забывать о порте. По умолчанию сервер использует порт 3306.

При изменении портов стоит учитывать, что некоторые порты (обычно до порта 9000) могут быть заняты, с подробным списком наиболее часто занятых портов можно ознакомится в интернете.

При изменении портов необходимо зайти в настройки и поменять на желаемый (рисунок 2.32).

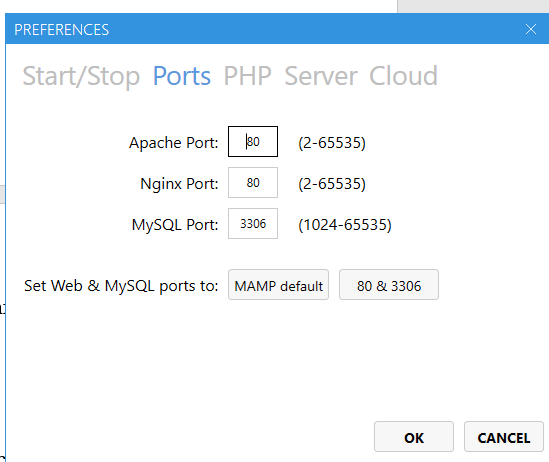


Рисунок 2.32 – Настройка портов

**2.6.1.4 Запуск и работа с сервером**

Для запуска системы обработки голоса достаточно написать одну команду «npm run dev» в папке BotTest\_V12 (рисунок 2.33).

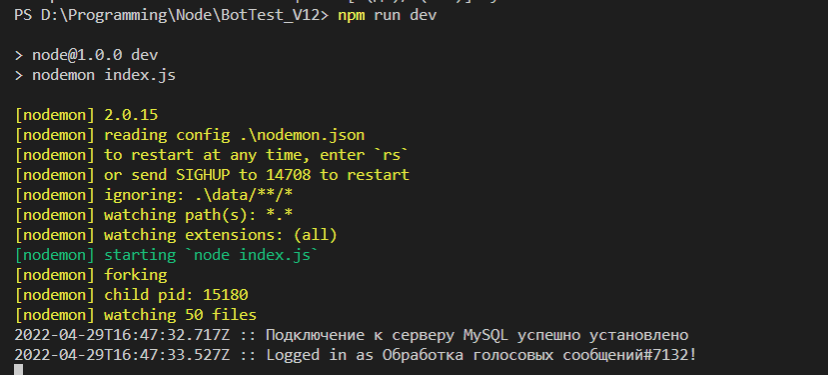


Рисунок 2.33 – Запуск системы

После запуска сервера будет выведена информация о потоке, количестве файлов, обработка ошибок в случае неудачно запуска как системы, так и БД.

Консоль имеет одну единственную команду «ctrl+c», при вводе которой сервер останавливает свою работу. При закрытии консоли сервер останавливает свою работу, поэтому необходимо, чтобы консоль была всегда открыта.

Следует учитывать, что для работы информационной системы работа сервера приложений необходима. Если сервер будет выключен при входе в систему будет выведено сообщение «Не удалось соединиться с сервером». Данная ошибка также может выводиться в случае, когда сервер запущен, но клиент настроен неправильно или не находится в той же локальной сети, что и сервер. Данная ошибка представлена на рисунке 2.34.

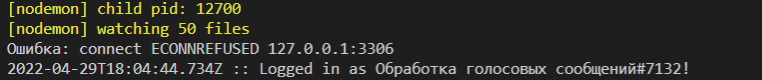


Рисунок 2.34 – Ошибка подключения

2.6.2 Клиентская часть

Данный подраздел описывает клиентскую часть ИС. Для её корректной работы необходимо учесть следующие минимальные системные и программные требования:

1. операционная система семейства Windows, начиная с 7 версии;
2. оперативная память от 512 MB;
3. свободное место на диске от 700 MB.

Соблюдение данных требований необходимо для работы клиента.

**2.6.2.1 Настройка клиента**

Для входа в систему необходимо использовать свои учетные данные, для регистрации на платформе Discord, данный процесс представлен на рисунке 2.35.

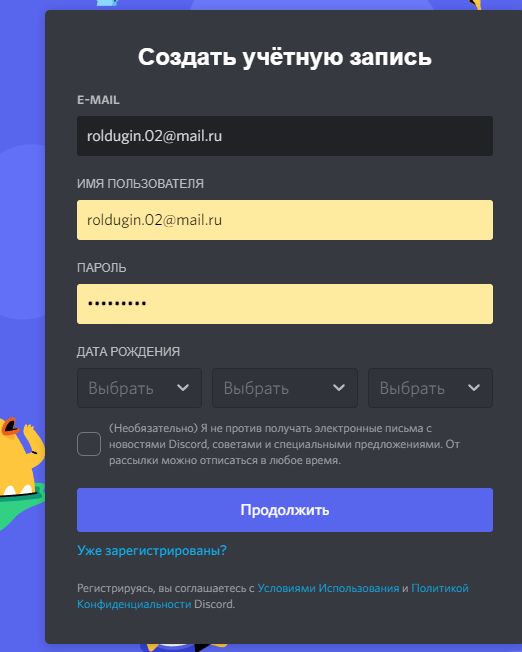


Рисунок 2.35 – Регистрация на платформе

После регистрации пользователя в системе, есть два варианта начать работу с ИС по обработке голосовых сообщений, такие как:

1. создать свой собственный сервер в Discord и добавить ИС на сам сервер;
2. присоединиться к уже существующему серверу, а также иметь нужные права доступа, чтобы добавить ИС.

Чтобы пригласить ИС, понадобится узнать CLIENT\_ID нашей ИС, сделать это можно перейдя по вкладкам OAuth2-General, и далее скопировать CLIENT\_ID, данный процесс представлен на рисунке 2.36.

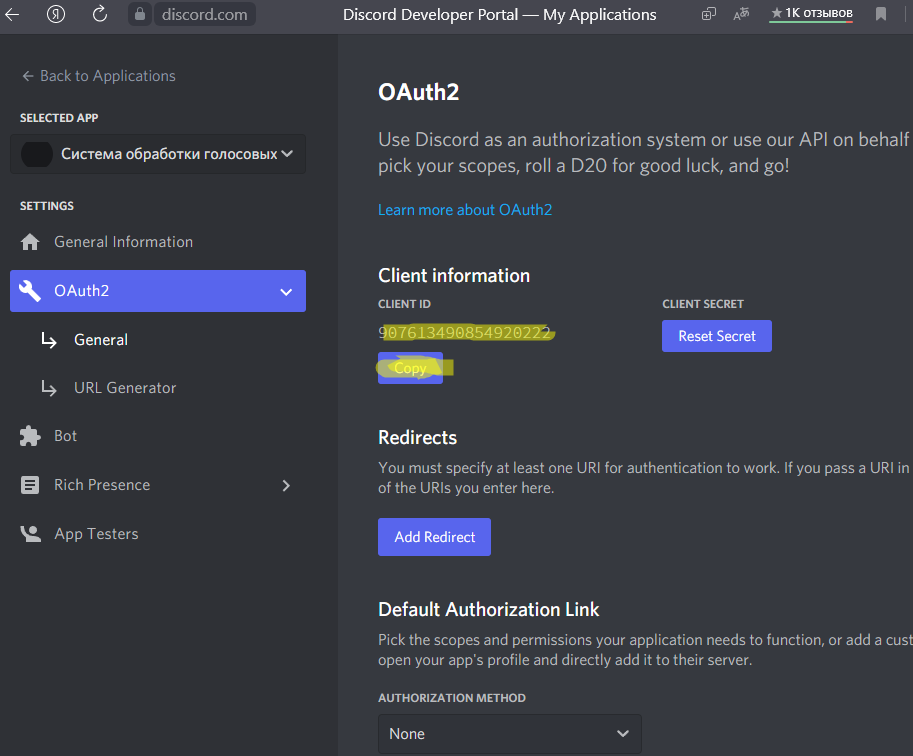


Рисунок 2.36 – Получение нужных данных

Чтобы добавить ИС на сервер необходимо быть авторизованным в Discrod и в браузере перейти по следующему адресу:

https://discordapp.com/oauth2/authorize?&client\_id="CLIENT\_ID"&scope=bot&permissions=0,

где CLIENT\_ID в данной ссылке необходимо заменить на номер ИС приложения. Вид добавления ИС на сервер представлен на рисунке 2.37.

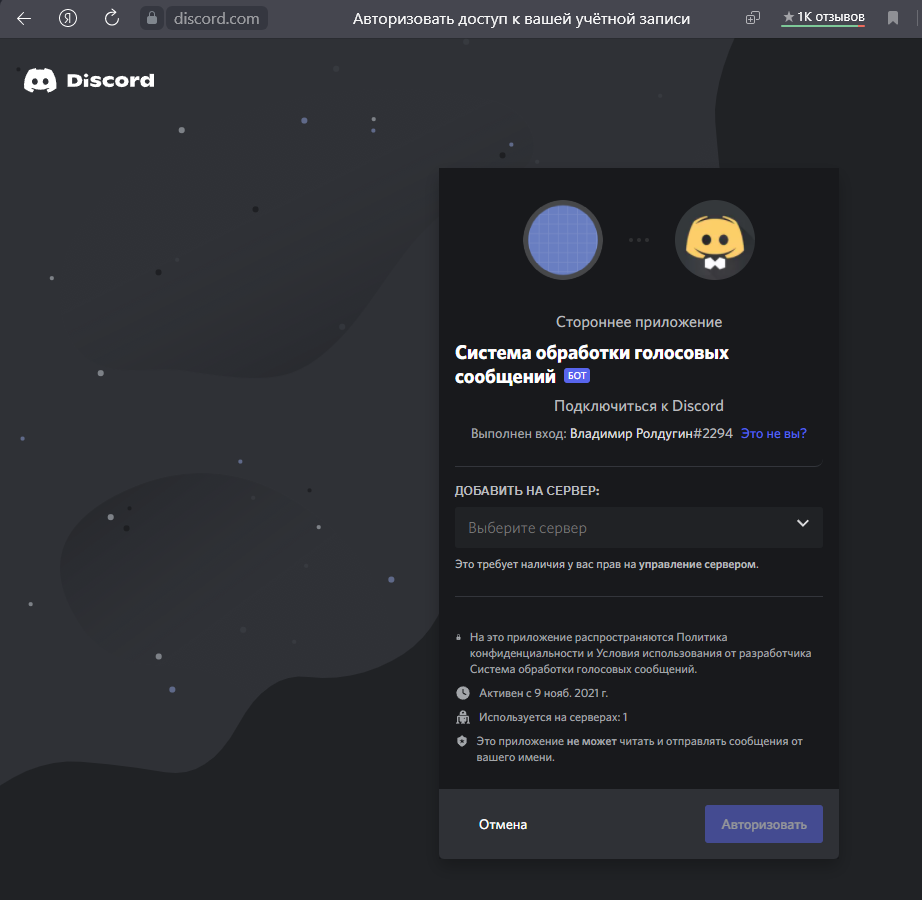


Рисунок 2.37 – Добавление системы на сервер

**2.6.3 Команда «!auth»**

Данная команда содержит функционал, связанный с авторизацией участников по их уникальному идентификатору на платформе Discrod. Она может быть использована в чате сервера. Вид команды представлен на рисунке 2.38.

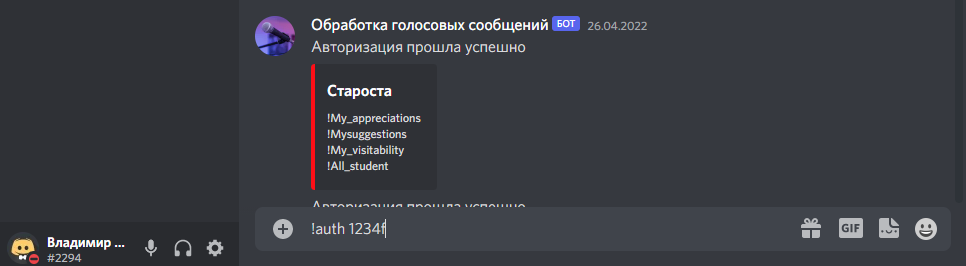


Рисунок 2.38 – Команда входа в систему

**2.6.4 Команда «!reg»**

Данная команда содержит функционал, связанный с регистрацией участников по их уникальному идентификатору на платформе Discrod. Она может быть использована в чате сервера. Вид команды представлен на рисунке 2.39.

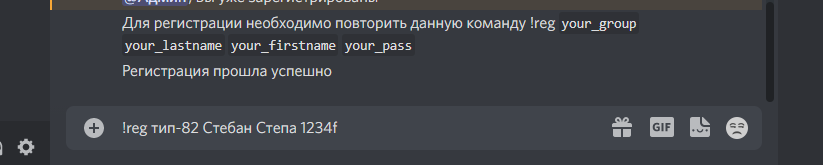


Рисунок 2.39 – Команда регистрации в системе

**2.6.5 Команда «!give\_elder»**

Данная команда содержит функционал, связанный с предоставлением новых прав студентам. Она может быть использована в чате сервера. Вид команды представлен на рисунке 2.40.



Рисунок 2.40 – Команда предоставления прав

**2.6.6 Команда «!join»**

Данная команда содержит функционал, связанный с записью конференции. После написания данной команды и успешной обработки ИС, в БД создается новая запись с конференцией и ее содержимым. Она может быть использована в чате сервера. Вид команды представлен на рисунке 2.41.

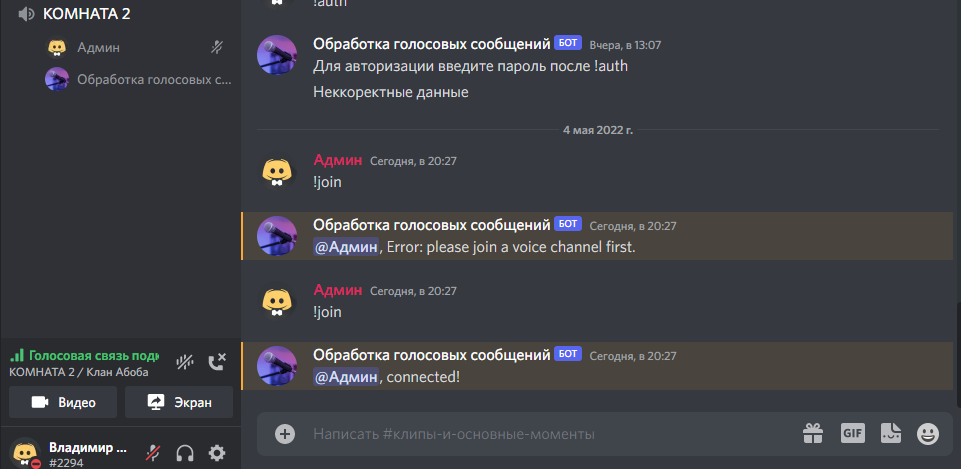


Рисунок 2.41 – Запись конференции

Между каналами можно свободно переключаться, при этом сама система останется там, куда ее пригласили в первый раз. Вид представлен на рисунке 2.42.

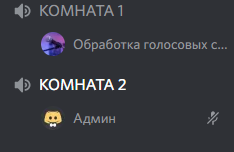


Рисунок 2.42 – Переключение между каналами

Использовать данную команду могут только пользователи с правами доступа, иначе система пришлёт сообщение «Пригласить или выгнать бота в конференцию может только староста или преподаватель». Вид об уведомлении представлен на рисунке 2.43.

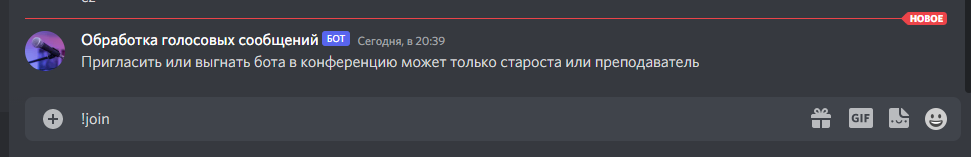


Рисунок 2.43 – Запись уже идёт

**2.6.7 Команда «!leave»**

Данная команда содержит функционал, связанный с завершением конференции. После написания данной команды и успешной обработки ИС, в БД складывается все время разговоров и запись прекращается. Она может быть использована в чате сервера. Вид команды представлен на рисунке 2.44.

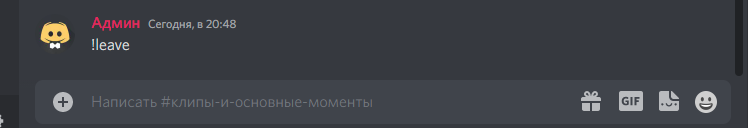


Рисунок 2.44 – Запись конференции

Завершить конференцию можно и находясь в другом канале, сделать это можно:

1. покинуть голосовой канал;
2. воспользоваться командой завершения конференции.

Пример представлен на рисунках 2.45.

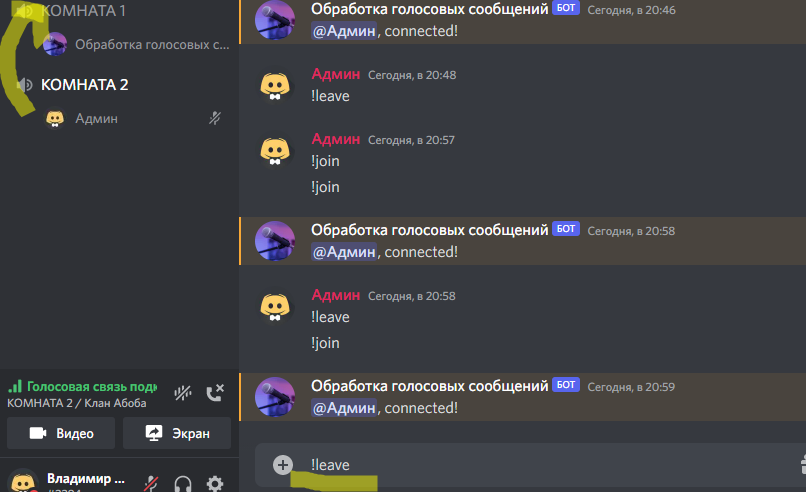


Рисунок 2.45 – Запись конференции

Если произойдет ситуация, когда пользователь завершит конференцию, но при этом, если до этого ИС не поставили на запись, то об этом система напишет в чат. Вид команды представлен на рисунке 2.46.



Рисунок 2.46 – Завершение не начатой конференции

**2.6.8 Команда «!all\_us»**

Данная команда содержит функционал, связанный с просмотром всех пользователей. Она может быть использована в чате сервера. Вид команды представлен на рисунке 2.47.

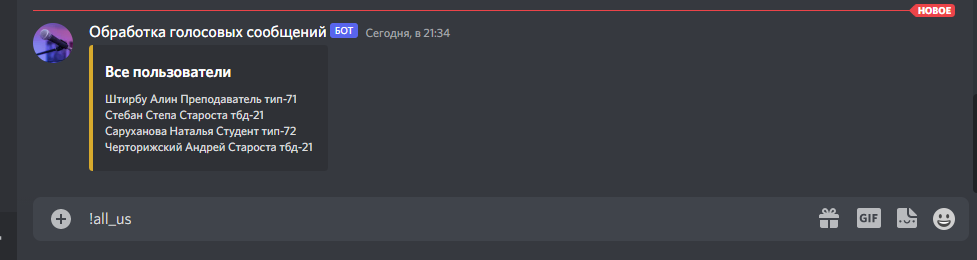


Рисунок 2.47 – Просмотр всех пользователей

Использовать данную команду могут только пользователи с имевшимися правами доступа, иначе система пришлёт сообщение «Только преподаватели могут смотреть всех студентов». Вид об уведомлении представлен на рисунке 2.48.

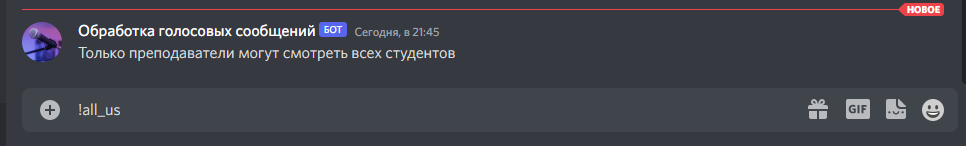


Рисунок 2.48 – Просмотр всех пользователей

**2.6.9 Команда «!students»**

Данная команда содержит функционал, позволяющий посмотреть студентов, которые находиться в той же группе, что и сам пользователь. Она может быть использована в чате сервера. Вид команды представлен на рисунке 2.49.

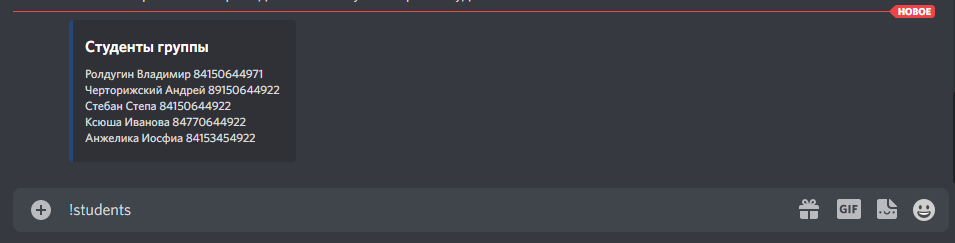


Рисунок 2.49 – Просмотр всех пользователей

Использовать данную команду могут только пользователи с имевшимися правами доступа, иначе система пришлёт сообщение «Только старосты или преподаватели могут смотреть студентов». Вид об уведомлении представлен на рисунке 2.50.

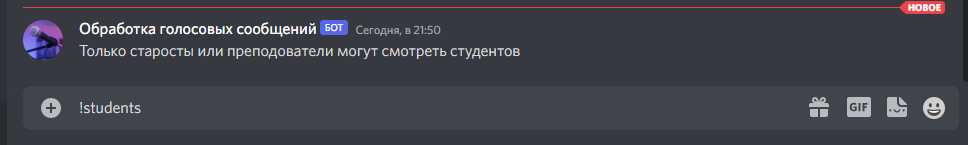


Рисунок 2.50 – Просмотр всех пользователей

**2.7 Вывод по разделу**

В данном разделе были рассмотрены разработка архитектуры информационной системы, разработка структуры данных, проектирование пользовательского интерфейса, описаны схемы алгоритмов основных функциональных возможностей ИС, отладка и тестирование программы и было описано полное руководство пользователя.

При разработке ИС стало необходимо добавить автоматическое завершении конференции, при бездействии или, когда пользователь забыл завершить запись, а также была добавлена возможность свободного перемещения по голосовым каналам.

Таким образом можно прийти к выводу, что поставленные задачи были решены в полном объёме.

**3 Охрана труда**

**3.1 Техника безопасности при работе на персональном компьютере**

Техника безопасности при работе с компьютером на предприятии описывает перечень правил и требований, что необходимо выполнить для безопасной работы с персональным компьютером (ПК). ПК является электроприбором, и как со всеми электроприборами, с ним нужно работать с осторожностью и соблюдая все требования техники безопасности.

К самостоятельной работе на ПК допускаются лица, прошедшие медицинский осмотр и не имеющие противопоказаний для работы на ПК, а также прошедшие инструктаж по охране труда и обучение безопасным методам выполнения работ.

Работник должен знать, что при выполнении работ на ПК на него могут оказывать воздействие следующие вредные и (или) опасные производственные факторы:

* + повышенная температура поверхностей ПК;
  + повышенная или пониженная температура воздуха рабочей зоны;
  + повышенное значение напряжения в электрической цепи, замыкание;
  + повышенный уровень статического электричества;
  + повышенный уровень электромагнитных излучений;
  + повышенная напряженность электрического поля;
  + отсутствие или недостаток естественного света;
  + недостаточная искусственная освещенность рабочей зоны;
  + повышенная яркость света;
  + повышенная контрастность;
  + зрительное напряжение;
  + длительные статические нагрузки;
  + монотонность трудового процесса.

Санитарно-эпидемиологические требования к эксплуатации ПК и организации рабочих мест с ПК определены СанПиН 2.2.272.4.1340-03.

Запрещается приступать к работе при обнаружении любой неисправности ПК и его составляющих, о неисправностях следует сообщить руководителю или лицам, осуществляющим техническое обслуживание оборудования.

При работе за ПК работник обязан содержать в порядке свое рабочее место и соблюдать дисциплину работы за ПК.

По окончании работы за ПК работник обязан:

* корректно закрыть все открытые программы, в том числе корректно завершить работу операционной системы;
* выключить питание системного блока, периферийных устройств, блока бесперебойного питания;
* осмотреть свое рабочее место и привести его в порядок.

**3.2 Требования к помещению (офиса, рабочего места)**

Помещение офиса, его размеры, должны соответствовать количественному составляющему штата сотрудников, а также размерам аппаратуры, используемой ими для работы.

Площадь рабочего места таких сотрудников не может быть менее 4,5 кв. м (если установлен плоский монитор) или менее 6 кв. м (если рабочее место оборудовано монитором старого типа, с кинескопом).

Требования к температуре и влажности воздуха, освещенности офисного помещения, а иногда даже к мебели жестко регламентированы.

В зависимости от среднесуточной температуры за окном определяется необходимая в помещении температура. При температуре за окном выше 10°С, в офисе должно быть по общему правилу 23-25°С, а при температуре ниже этого порога - 22-24°.

Помещение должно быть оснащено естественным освещением, что обеспечивает зрительный комфорт, и снижает зрительную и общую утомленность.

Помещение должно снижать шум, проникающий извне, а также следует использовать специальные приспособления для уменьшения количества шума, созданного используемой при работе аппаратурой. Снижение шума является важным фактором для улучшения условий работы.

Помещение должно быть оснащено кондиционером, что обеспечивает необходимый для работы микроклимат на протяжении всех сезонов и очистку воздуха от пыли и других веществ, находящихся в воздухе.

**3.3 Мероприятия по противопожарной технике**

Формирование перечня мероприятий по пожарной безопасности офиса зависит от нескольких факторов: места размещения помещения в здании, размеров офиса, числа сотрудников, количества электроприборов, наличия складов и производственных материалов, находящихся рядом с офисом.

Действующее законодательство выдвигает следующие требования к пожарной безопасности офисных помещений:

* наличие в специально отведенных местах информационных табличек с указателями и номерами телефонов спасательных служб;
* разработка для каждого этажа индивидуальных планов эвакуации;
* наличие первичных средств пожаротушения, противопожарной сигнализации, автономных систем борьбы с огнем;
* разработка правил техники пожарной безопасности в офисе, обязательное проведение инструктажа с сотрудниками.

Владелец офиса или его арендатор обязан назначить ответственного за пожарную безопасность объекта. На этого человека подкладывается контроль выполнения норм законодательства всеми сотрудниками и проведение периодических учебных семинаров и инструктажей.

Каждый сотрудник должен знать и выполнять в обязательном порядке предусмотренные меры пожарной безопасности в офисе. Инструкции и обучение должны обеспечить в сотрудников знаниями о том:

* как действовать в случае возникновения пожара;
* как работают первичные средства пожаротушения;
* куда обращаться за помощью;
* где находятся эвакуационные пути и как быстро покинуть горящее помещение;
* как выполняется первая медицинская помощь при ожогах и отравлении газообразными продуктами горения.

Одним из требований, которые предусматривает организация пожарной безопасности в офисе, является наличие противопожарных сигнализаций и систем предупреждения.

Чтобы гарантировать защиту людей и материальных ценностей сигнализация должна отвечать следующим требованиям:

* быть отказоустойчивой и надежной работе независимо от условий, включая и экстремальные условия эксплуатации;
* обеспечивать большую зону покрытия – датчики должны контролировать все места возможного возгорания, не оставляя «слепых зон»;
* гарантировать непрерывную работу в режиме 24/7 даже в тех случаях, когда не функционирует основная система электроснабжения.

**4 Технико – экономическая часть**

**4.1 Технико – экономические расчеты экономической целесообразности разработки программы**

Задача состоит в разработке системы обработки голосовых сообщений в текстовый формат на платформе Discrod, а также команды для работы с БД.

Система была разработана для хранения и автоматизации работы с данными, с целью повышения удобства работы и улучшения производительности труда.

Система разработана с использованием языка JavaScript, платформы Node и БД MySQL.

Для того, чтобы говорить об экономической эффективности разработанного проекта, надо рассчитать его себестоимость.

Себестоимость – это затраты предприятий, связанные с производством и реализацией продукции. Калькуляция себестоимости выпускаемой продукции включает в себя затраты на сырье и материалы, оплату труда, страховые взносы, амортизацию основных фондов и прочие расходы. Расчет себестоимости показан на формуле (4.1).

|  |  |
| --- | --- |
| С = М + КИ(ПФ) + Тр−з + Зосн + Здоп + Св + Нцех + Нзав + Впр, | (4.1) |

где С – себестоимость, руб.;

М – стоимость материалов, основы для полуфабрикатов, разделенных на вспомогательные (не входящие в состав готовой продукции) и основные, руб.;

КИ(ПФ) – стоимость комплектующих материалов, готовых элементов конечного продукта, используемых для сборки, ремонта и упаковки, руб.;

Тр-з – транспортно-заготовительные расходы, связанные с заготовкой, доставкой и хранением материальных ценностей (товаров, сырья, материалов, инструментов), руб.;

Зосн – заработная плата основная основных производственных рабочих, определяется в зависимости от трудоемкости, сложности и действующей формы оплаты труда, руб.;

Здоп – заработная плата дополнительная основных производственных рабочих, расходуется на оплату отпусков, отдельных видов премирования, пособия для уходящим в армию, выплат государственных и общественных обязанностей, руб.;

Св – страховые взносы основных производственных рабочих, расходуется на пенсии и медицину, руб.;

Нцех – накладные расходы цеховые, расходуются на амортизацию оборудования цеха, заработную плату всех работников цеха (исключая основных производственных рабочих), ЖКХ, текущий и капитальный ремонт помещений цеха, руб.;

Нзав – накладные расходы завода, расходуются на амортизацию общезаводского оборудования, заработную плату всех работников завода (исключая работников цехов), ЖКХ, текущий и капитальный ремонт здания завода, аренду, отчисления вышестоящим организациям, руб.;

Впр – внепроизводственные расходы, расходуются рекламу, упаковку, командировки, презентации новых разработок, руб.

Ввиду специфики работы программиста и отсутствия материальной части формула приобретает вид (4.2):

|  |  |
| --- | --- |
| С = Зосн + Здоп + Св + Нцех + Нзав + Впр, | (4.2) |

где Зосн – заработная плата основная программистов, определяется в зависимости от трудоемкости задачи и оклада, руб.;

Здоп – заработная плата дополнительная программистов, расходуется на оплату отпусков, отдельных видов премирования, пособия для уходящим в армию, выплат государственных и общественных обязанностей, руб.;

Св – страховые взносы программистов, расходуется на пенсии и медицину, руб.;

Нцех – накладные расходы отдела, расходуются на амортизацию компьютеров и другого оборудования, заработную плату всех работников отдела (исключая программистов), ЖКХ, текущий и капитальный ремонт помещений отдела, руб.;

Нзав – накладные расходы фирмы, расходуются на амортизацию общефирменного оборудования, заработную плату всех работников фирмы (исключая работников отделов), ЖКХ, текущий и капитальный ремонт здания фирмы, аренду, отчисления вышестоящим организациям, руб.;

Впр – внепроизводственные расходы, расходуются на антивирусные программы, оплату интернета, покупку нового оборудования, сопровождение программы во время ее выполнения, руб.

В связи с высокой стоимостью оборудования для программиста, необходимостью быстрой амортизации и большим объемом потребляемой электроэнергии формула приобретает вид (4.3):

|  |  |
| --- | --- |
| С = Зосн + Здоп + Св + А + Э + Нцех + Нзав + Впр, | (4.3) |

где А – сумма амортизации оборудования, руб.;

Э – плата за потребленную электроэнергию, руб.

**4.2 Расчетная часть**

**4.2.1 Расчет трудоемкости разработанной программы**

Для определения себестоимости решения задачи необходимо, прежде всего, найти трудоемкость решения задачи.

Трудоемкость – это сумма затрат труда (по времени), необходимых для изготовления единицы продукции, которая предназначена для решения разработки программы в соответствии с содержанием задания.

Трудоемкость рассматривается как сумма затрат времени на разных этапах решения задачи. Общая трудоемкость рассчитывается по формуле (4.4).

|  |  |
| --- | --- |
| То = Ти + Та + Тбс + Тп + Тот + Тд + Тм-р + Тэвм, | (4.4) |

где Tо – трудоемкость общая, час;

Ти – затраты труда на изучение материала, описание задачи, час;

Та – затраты труда на разработку алгоритмов решения задачи, час;

Тбс – затраты труда на разработку блок-схем алгоритма программы, час;

Тп – затраты труда на программирование, час;

Тот – затраты труда на отладку программы, час;

Тд – затраты на оформление документации, час;

Тм-р – затраты труда на машинно-ручные работы, час;

Тэвм – время машинного счета на ЭВМ, час.

Слагаемые трудоемкости определяются через количество программных команд данной стадии разработки.

Затраты труда на изучение и описание задачи определяются по формуле (4.5).

|  |  |
| --- | --- |
|  | (4.5) |

где Q – предполагаемое число программных команд данной стадии разработки;

β – коэффициент, учитывающий качество описания задачи, равный 1,3 по данным предприятия;

В – скорость программиста, количество команд в час;

Ккв – коэффициент квалификации исполнителя, равный 0,8 по данным предприятия.

**Расчет трудоемкости на этапе изучения и описания задачи**

Число программных команд данной программы равно 1323 командам.

По формуле (3.5) трудоемкость изучения материала и описания задачи.

B = 89;

β = 1,3;

Ккв = 0,8.

**Расчет затрат труда на этапе разработки алгоритма решения задачи**

Величина Ta находится по формуле, идентичной Tи.

B = 26;

β = 1,3;

Ккв = 0,8.

**Расчет затрат труда на этапе разработки схем алгоритма программы**

Величина Tбс находится по формуле, идентичной Tи.

B = 16;

β = 1,3;

Ккв = 0,8.

**Расчет затрат труда на этапе программирования**

Величина Tп находится по формуле, идентичной Ти.

B = 26;

β = 1,3;

Ккв = 0,8.

**Расчет затрат труда на этапе отладки**

Величина Tот находится по формуле, идентичной Tи.

B = 30;

β = 1,3;

Ккв = 0,8.

**Расчет затрат труда на этапе документации**

Величина Tд находится по формуле, идентичной Tи.

B = 21;

β = 1,3;

Ккв = 0,8.

**Расчет затрат труда на этапе машинно-ручных операций**

Трудоемкость на этапе машинно-ручных операций рассчитывается по формуле (4.6).

|  |  |
| --- | --- |
|  | (4.6) |

где t – время ввода информации, сек.

Время ввода информации находится по формуле (4.7).

|  |  |
| --- | --- |
|  | (4.7) |

где С – объем программы в символах.

По формуле (4.7) необходимо найти время ввода информации при С = 29372.

Зная время ввода информации, можно найти трудоемкость на этапе машинно- ручных операций по формуле (4.6).

**Расчет затрат труда на этапе машинного времени**

Время машинного счета на ЭВМ вычисляется по формуле (4.8).

|  |  |
| --- | --- |
|  | (4.8) |

где tвв – время ввода, час;

tвыв – время вывода, час;

tсч – время счета, час.

По статистическим данным Тэвм = 0,02 часа.

**Расчет общей трудоемкости**

Зная все затраты труда на всех этапах решения задачи, можно вычислить общую трудоемкость решаемой задачи по формуле (4.4).

То = 24,16+82,69+134,37+82,69+71,66+102,39+3,05+0,02=  
= 501,03

Трудоемкость решения задачи на ЭВМ составляет 501,03 часа.

**4.2.2 Расчет себестоимости разработанной программы**

Для нахождения себестоимости разработки программы при решении задачи на ЭВМ необходимо использовать формулу (4.9).

|  |  |
| --- | --- |
|  | (4.9) |

**Расчет основной заработной платы**

Заработная плата – это часть национального дохода, переданного в личное распоряжение работника в соответствии с количеством и качеством затраченного им труда. Заработная плата состоит из основной, дополнительной заработной платы и страховых взносов.

Основная заработная плата зависит от степени квалификации работника, уровня сложности, количества и качества выполняемой работы, а также условий, в которых выполняется работа и определяется по формуле (4.10).

|  |  |
| --- | --- |
|  | (4.10) |

где Сч – часовая тарифная ставка, руб.;

То – общая трудоемкость решения задачи, час.

Часовая тарифная ставка определяется по формуле (4.11).

|  |  |
| --- | --- |
|  | (4.11) |

где Оклад – месячный штатный оклад программиста по данным предприятия, равен 45 000 рублей;

22,8 – среднее количество рабочих дней в месяц;

8 – количество рабочих часов в смену.

Найдем часовую тарифную ставку по формуле (4.11).

Для нахождения основной заработной платы необходимо использовать формулу (4.10).

**Расчет дополнительной заработной платы**

Далее, необходимо найти дополнительную заработную плату по формуле (3.12), которая равна 80% от основной заработной платы по данным предприятия.

|  |  |
| --- | --- |
|  | (4.12) |

**Расчет страховых взносов**

Зная размер основной и заработной платы, можно вычислить сумму страховых взносов, которая составляет 30,2% от размера фонда оплаты труда (ФОТ) и является обязательным государственным налогом. ФОТ рассчитывается по формуле (4.13).

|  |  |
| --- | --- |
|  | (4.13) |

Зная ФОТ можно рассчитать страховые взносы по формуле (4.14).

|  |  |
| --- | --- |
|  | (4.14) |

**Расчет амортизационных отчислений**

Амортизация – это ежемесячное денежное отчисление для возмещения износа оборудования. Амортизация рассматриваемой программы вычисляется в три этапа. Сначала рассчитывается годовая амортизация по формуле (4.15).

|  |  |
| --- | --- |
|  | (4.15) |

где Сперв – первоначальная себестоимость оборудования, использованного для разработки программы, по данным предприятия 80 000 руб.;

4 – срок окупаемости оборудования, год.

Далее рассчитывается амортизация дневная по формуле (4.16).

|  |  |
| --- | --- |
|  | (4.16) |

где 247 – количество рабочих дней в 2022 году.

Чтобы найти амортизацию программы, необходимо вычислить амортизацию часовую, исходя из формулы (4.17).

|  |  |
| --- | --- |
|  | (4.17) |

где 8 – количество рабочих часов в смену.

Итого на каждый час работы программиста приходится 10.12 рубля амортизации. Вычислим амортизацию рассматриваемой программы по формуле (4.18).

|  |  |
| --- | --- |
|  | (4.18) |

**Расчет затрат на электроэнергию**

Затраты на электроэнергию рассчитываются по формуле (4.19).

|  |  |
| --- | --- |
|  | (4.19) |

где Тариф – действующий тариф на электроэнергию в Москве, по данным предприятия 6,17 руб/кВт–час.

**Расчет накладных цеховых расходов**

Накладные расходы отдела определяются в процентном отношении от основной зарплаты, составляют 200% по данным предприятия. В состав накладных расходов отдела включаются такие затраты как заработная плата аппарата управления отдела (начальника отдела, заместителя начальника, системного администратора), амортизационные отчисления на текущий и капитальный ремонт заданий и оборудования, сооружений, на охрану труда в данном отделе и на непроизводительные затраты. Рассчитывается по формуле (4.20).

|  |  |
| --- | --- |
|  | (4.20) |

**Расчет себестоимости цеховой**

Себестоимость отдела – это показатель затратности производства, его анализ позволяет выявить фактор отклонения от процентного объема расходов и сравнить эффективность работы разных отделов. Рассчитывается по формуле (4.21).

|  |  |
| --- | --- |
| Сцех = Зосн + Здоп + Св + А + Э + Нцех , | (4.21) |

**Расчет накладных заводских расходов**

Накладные расходы фирмы определяются в процентном отношении от основной зарплаты и составляют 100% по данным предприятия. Накладные расходы фирмы – это расходы по управлению фирмой, содержание общефирменного персонала с отчислением на страховые взносы, расходы по командировкам, амортизационные отчисления общефирменного оборудования, на текущий и капитальный ремонт зданий, отчисление вышестоящим организациям. Рассчитываются по формуле (4.22).

|  |  |
| --- | --- |
| Нзав = Зосн × 100%, | (4.22) |

**Расчет себестоимости производственной**

Производственная себестоимость определяется путем суммирования общезаводских и целевых расходов с себестоимостью цеховой. Она включает производственные затраты всех отделов, занятых производством продукции или услуг, и расходы по общему управлению предприятием. Себестоимость производственная рассчитывается по формуле (4.23).

|  |  |
| --- | --- |
| Спр = Сцех + Нзав, | (4.23) |

**Расчет внепроизводственных расходов**

Внепроизводственные расходы являются неотъемлемой частью затрат предприятия и могут включать в себя рекламу, сопровождение программы во время ее исполнения на предприятии, оплату интернета, первоначальную настройку оборудования и сетей, поддержание работоспособности системы ЭВМ. По данным предприятия они составляют 10% от стоимости заводской и рассчитываются по формуле (4.24).

|  |  |
| --- | --- |
| Впр = Спр × 10%, | (4.24) |

**Расчет полной себестоимости решения задачи**

Таким образом, зная все необходимые величины, можно вычислить полную себестоимость. Она включает затраты организации не только на выпуск продукции и организацию производственного процесса, но и на ее реализацию, иначе – на ее поставку на рынок конечного товара и услуг. Она учитывается при формировании цены реализации этой продукции и служит показателем для определения суммы прибыли, получаемой от ее продажи. Вычисляется по формуле (4.25).

|  |  |
| --- | --- |
|  | (4.25) |

Себестоимость решения задачи на ЭВМ равна руб.

**4.2.3 Анализ возможных путей снижения себестоимости**

**4.3 Графическая часть**

В таблице 4.1 представлена производительность труда программиста.

Таблица 4.1 – Производительность труда программиста

|  |  |
| --- | --- |
| Характер работы | Производительность,  количество команд/час |
| Изучение описания задачи | 24 |
| Разработка алгоритмов решения | 82 |
| Разработка схем алгоритма | 134 |
| Программирование по готовой схеме алгоритма с использованием алгоритмического языка | 82 |
| Автономная отладка программы | 429 |
| Оформление документации | 102 |

В таблице 4.2 представлена трудоемкость при решении задачи на ЭВМ.

Таблица 4.2 – Трудоемкость при решении задачи на ЭВМ

|  |  |  |
| --- | --- | --- |
| Наименование затрат | Единица измерения | Трудоемкость в часах |
| Трудоемкость решения задачи на ЭВМ | час | 501,03 |
| В том числе: | | |
| Время изучения описания задачи | час | 24,16 |
| Время на разработку алгоритма задачи | час | 82,69 |
| Время на разработку схемы алгоритма | час | 134,37 |
| Время на программирование | час | 82,69 |
| Время на отладку программы | час | 71,66 |
| Время на оформление документации | час | 102,38 |
| Время на машинно-ручные работы | час | 3,05 |
| Время машинного счета | час | 0,02 |

В таблице 4.3 представлена калькуляция разработки программы.

Таблица 4.3 – Таблица калькуляции

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование статьи калькуляции | Сумма в руб. | Процент к  итогу | Процент к  основной  зарплате |
| Прямые затраты: | | | |
| Основная заработная плата | 123609,11 | 9,4 | 100,00 |
| Дополнительная заработная плата | 98887,28 | 7,52 | 80,00 |
| Страховые взносы | 67193,91 | 9,4 | 54,36 |
| Амортизация оборудования | 9061,91 | 0,69 | 7,33 |
| Затраты на электроэнергию | 3091,35 | 0,24 | 2,5 |
| Косвенные затраты: | | | |
| Накладные цеховые расходы |  | 18,81 | 200,0 |
| Себестоимость цеховая |  | – | – |

Продолжение таблицы 4.3

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование статьи калькуляции | Сумма в руб. | Процент к  итогу | Процент к  основной  зарплате |
| Накладные заводские расходы |  | 9,4 | 100,0 |
| Себестоимость производственная |  | – | – |
| Внепроизводственные расходы |  | 9,1 | 54,46 |
| Себестоимость полная | 1 314 392,93 | 100,0 | 1063,35 |

В таблице 4.4 представлены технико-экономические показатели.

Таблица 4.4 – Технико-экономические показатели

|  |  |  |  |
| --- | --- | --- | --- |
| Технико- экономический показатель | Единица измерения | Формула | Результат |
| Трудоёмкость решения задачи на ЭВМ | час | То = Ти + Та + Тбс + Тп +  + Тот + Тд + Тм-р + Тэвм | 501,03 |
| Себестоимость решения задачи на ЭВМ | руб. | С = Зосн + Здоп + Св + А +  +Э + Нцех + Нзав + Впр |  |

**ЗАКЛЮЧЕНИЕ**

В результате данной дипломной работы была реализована система обработки голосовых сообщений в текстовый формат на платформе DISCROD.Реализовано управление системой с помощью специальных команд в чате сервера. Для сохранения конфиденциальности все сообщения отправляться в личные сообщения, что не позволяет другим пользователям отследить историю всех действий сторонних пользователей. В систему была добавлена возможность добавления роли выбранному пользователю, что позволяет разграничить доступ к самой системе

Был проведен анализ литературных источников, который показал, что в настоящее время существуют различные методы и алгоритмы распознавания речи, однако, все они не лишены недостатков.

Для разработки системы была изучена платформа NODE и использован принцип разделения программы на модули, что поспособствовало легче ориентироваться в программе. Кроме того, были улучшены навыки в написании кода на языке программирования JavaScript. Таким образом дипломный проект послужил хорошей практической задачей для закрепления ново изученного материала.

Программа структурирована в соответствии с принципом разделения программы на модули, в листинге приведен код с комментариями.

В ходе написания дипломного проекта были отточены навыки разработки, отладки, тестирования и документировании.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Алексей Васильев JavaScript в примерах и задачах,715 стр, 2017.
2. Джон Дакетт JavaScript и Query,629 стр, 2015.
3. Херрон Девид Node.js. Разработка на JavaScript,144 стр,2016.
4. Емануел ДелБоно Разработка NODE.JS BASICS,121 стр, 2016.
5. Майк Кантелон Node.js в действии,548 стр,2014.
6. Маниел Кеслинг NODE.JS NOTES FOR PROGESSIONALS,125 стр, 2017.
7. Дуглас Крокфорд Как устроен JavaScript 304 стр,2019.
8. Хавербеке Марейн JavaScript. веб-программирование,479 стр., 2019.
9. Осипов Д.Л Технологии проектирования баз данных – ДМК–Пресс, 499 стр., 2019.
10. Шелли Пауэрс Node. Переходим на сторону сервера,304 стр,2017.
11. Сесил Р. Чистая архитектура – Питер,352 стр, 2019.
12. Фаулер М. Рефакторинг. Улучшение существующего кода,429 стр., 2008.
13. Минник, Холланд JavaScript для чайников,321 стр,2017.
14. Документация по языку программирования JS //URL:  
    <https://developer.mozila.org/ru/docs/Web/JavaScript>
15. Документация по NODE JS //URL: [https:/node.org/ru/docs/](https://stackoverflow.com/)
16. Форум программистов //URL: <https://stackoverflow.com/> (дата обращения: 30.05.2022)
17. Документация MySQL //URL: <https://dev.mysql.com/doc> (дата обращения: 30.05.2022)
18. Документация по DISCROD JS //URL: https:/discord.js.org/
19. Портал статей, связанных с дизайном интерфейса //URL: <https://infogra.ru> (дата обращения: 30.05.2022)
20. Портал статей, связанных с разработкой //URL: <https://habr.com/> (дата обращения: 30.05.2022)
21. Портал уроков по языкам программирования //URL: <https://itproger.com> (дата обращения: 30.05.2022)
22. Форум по языку программирования JS //URL:  
    <https://www.valentinog.com/blog/jsdoc/>
23. Форум программистов //URL: <https://cyberforum.ru/> (дата обращения: 30.05.2022)
24. Форум программистов //URL: <https://overcoder.net/> (дата обращения: 30.05.2022)
25. Пакеты для проекта NPM //URL: <https://www.npmjs.com/package/discrod.js>

ПРИЛОЖЕНИЕ А

(обязательное)

Листинги программы

**A.1 Листинги команд приложение**

**A.1.1 Листинг команды Reg**

/\*

\* Дипломный проект по теме

\* "Разработка сервиса анализа и обработки голосовых сообщений в дискорд

\*

\* Разработал: Ролдугин Владимир Дмитриевич, группа ТИП - 82

\* Специальность: 09.02.07

\* Дата и номер версии: 09.04.2020 v1.1

\* Язык программирования: JavaScript

\* Среда разработки: Microsoft Visual Studio Code 2015

\*

\* Задание:

\* Разработать систему преобразования аудио сообщений в текстовый формат. Система

\* должна хранить и предоставлять интерфейс управления данными.

\*/

var con = require('../DB/conDb');

module.exports = async (bot,message,args,argsF) => {

const cont = argsF.join(" ") // все что пишеться после !reg

const res1 = cont.split(" ") // Разделеяем все записи на масив

function Check\_group(callback) {

//Проверка существует ли данная группа

con.query('SELECT \* FROM `groups` WHERE `groups`.`name` =?',

[res1[0]], async (err,res,fields)=>{

//Проверка на выполнение запроса

if(err){

console.log(err.message);

return callback(0,false) }

if(res.length ==0) {

message.channel.send("такой группы не существует")

return callback(0,false)

}else{

// берем id группы если такая существует

return callback(res[0].id\_g,true)

}

})

}

function Check\_student(grop,callback) {

// Проверка сушествует ли Фамилия и Имя студента

con.query('SELECT \* FROM `students` WHERE `students`.`lastname` = ? AND `students`.`firstname` =? AND `students`.`id\_g` = ?',

[res1[1],res1[2],grop], async (err,res,fields)=>{

//Проверка на выполнение запроса

if(err){

console.log(err.message);

return callback(0,false)}

if(res.length ==0) {

// console.log(res1[0],res1[1],res1[2]);

message.channel.send("такого студента не существует ")

return callback(0,false)

}

else{

return callback(res[0].id\_s,true)

}

})}

function Insert\_reg(student) {

console.log(student);

//Строка для добавления пользователя

con.query('INSERT INTO `users` (`id`, `id\_s`, `pass`, `id\_r`) VALUES (?, ?, ?, ?)',

[message.author.id,student,res1[3],1], async (err,res,fields)=>{

if(err)

return console.log(err.message);

message.delete()

message.channel.send("Регистрация прошла успешно")

var role = message.guild.roles.cache.find(role => role.id === "931628294409945138")

var user = message.guild.members.cache.get(message.author.id);

user.roles.add(role) })

//Завершение строки добавления

}

function regUser() {

result = true

grop = 0

// Вызываем метод для проверки на сущ группы, если результат true,

// группа найдена, если нет, то выходим и говорим "Регистрация не была произведена"

//Так же если группа найдена , то в параметр груп записываем id группы

Check\_group(function (grop,result) {

console.log(result);

// Если группа найдена

if(result){

//Тут проблема в том, что параметр !!!!grop!!!! из "Check\_group" я не могу вытащить отсюда

//Я создаю новую переменную, и присваиваю туда grop, но когда выхожу почему-то не меняет!!

// Поэтому в этой проверке я вызываю проверку на студента, туда я передаю группу, и в запросе

// смотрю есть ли такой студент или нет

// так же есть параметр student, это id студента, который ищется по фамилии и его имени, и найденому id группы,

// после чего Insert\_reg выполняет запрос на добавление юзера

Check\_student(grop,function (student,result) {

console.log(result);

if(result){

Insert\_reg(student) }

else

return console.log("Регистрация не была произведена");}) }

// Если группа не найдена

else

return console.log("Регистрация не была произведена"); })}

con.query('SELECT \* FROM `users` WHERE `users`.`id` = ? ',

[message.author.id], async (err,res,fields)=>{

if(err)

return console.log(err.message);

if(!res.length ==0) {

message.delete()

return message.reply("Вы уже зарегистрированы")}

else{

if(res1[0]==null || res1[1]==null||res1[2]==null||res1[3]==null){

message.delete()

return message.channel.send("Для регистрации необходимо повторить данную команду !reg `your\_group` `your\_lastname` `your\_firstname` `your\_pass`")

}

regUser() } })};

**A.1.2 Листинг команды Students**

/\*

\* Модуль: Students

\* Цель: Команда, позволяющая показать студентов своей группы

\*/

var con = require('../DB/conDb');

module.exports = async (message,Discord) => {

var user = message.guild.members.cache.get(message.author.id);

// 'SELECT `students`.`id\_g` FROM `users`,`students` WHERE `users`.`id\_s`=`students`.`id\_s` AND `users`.`id` = ?'

function get\_idG(callback) {

//Проверка сушествует ли данная группа

con.query('SELECT `students`.`id\_g` FROM `users`,`students` WHERE `users`.`id\_s`=`students`.`id\_s` AND `users`.`id` = ?',

[message.author.id], async (err,res,fields)=>{

//Проверка на выполнение запроса

if(err){

console.log(err.message);

return callback(0,false)

}

if(res.length ==0) {

message.channel.send("такой группы не существует")

return callback(0,false)

}else{

// берем id группы если такая существует

return callback(res[0].id\_g,true) } }) }

str =""

id\_gg=0

get\_idG(function (grop,result) {

if(user.roles.cache.find(role => role.name == "Преподаватель")||user.roles.cache.find(role => role.name == "Староста")){

con.query('SELECT \* FROM `students` WHERE `students`.`id\_g`=?',

[grop], async (err,res,fields)=>{

if(err)

return console.log(err.message);

if(!res.length ==0) {

message.delete()

for (let i = 0; i < res.length; i++) {

str+=res[i].lastname+" "+res[i].firstname+" "+res[i].phone+"\n"}

const embed = new Discord.MessageEmbed()

.setColor('#22467d')

.setTitle('Студенты группы')

.setFooter(str)

return user.send(embed); }

else{

message.delete()

return user.send("Студентов нет") }}) }

else{

message.delete()

return user.send("Только старосты или преподователи могут смотреть студентов") })}

**A.1.3 Листинг команды Join\_leave**

/\*

\* Модуль: Join\_leave

\* Цель: Команда, пригласить или выгнать систему из голосового канала

\*/

var con = require('../DB/conDb');

module.exports = async (msg,guildMap,connect) => {

var user = msg.guild.members.cache.get(msg.author.id)

if(user.roles.cache.find(role => role.name == "Преподаватель") || user.roles.cache.find(role => role.name == "Староста")){

try {

if (!('guild' in msg) || !msg.guild) return; // prevent private messages to bot

const mapKey = msg.guild.id;

global.Map\_Key\_global = mapKey

global.user\_id = msg.author.id

if (msg.content.trim().toLowerCase() == '!join') {

if (!msg.member.voice.channelID) {

msg.reply('Error: please join a voice channel first.')} else {

if (!guildMap.has(mapKey)){

con.query("INSERT INTO `records` (`id`, `speech`, `duration`) VALUES (NULL, '', '0.0')",

[], async (err,fields)=>{

if(err)

return console.log(err.message); })

con.query('SELECT `records`.`id` FROM `records` ORDER BY `records`.`id` DESC LIMIT 1',

[], async (err,res,fields)=>{

if(err)

return console.log(err.message);

if(res.length ==0)

return console.log("Нету записей");

await connect(msg, mapKey,res[0].id)}) } else

msg.reply('Already connected')} }

else if (msg.content.trim().toLowerCase() == '!leave')

require('../global\_fun/leave')(guildMap,mapKey,msg)} catch (e) {

console.log('discordClient message: ' + e)

msg.reply('Error#180: Something went wrong, try again or contact the developers if this keeps happening.');}}

else

user.send('Пригласить или выгнать бота в конференцию может только староста или преподаватель')

}

**A.1.4 Листинги команды Auth**

/\*

\* Модуль: Auth

\* Цель: Команда, позволяющая авторизовать пользователя в системе

\*/

var con = require('../DB/conDb');

module.exports = async (bot,message,args,argsF,Discord) => {

const cont = argsF.join(" ")

if(cont==""){

message.delete()

return message.channel.send("Для авторизации введите пароль после !auth")

}

con.query('SELECT \* FROM `users` WHERE `users`.`id` = ? AND `users`.`pass` = ? ',

[message.author.id, cont], async (err,res,fields)=>{

if(err)

return console.log(err.message);

if(!res.length ==0) {

// setTimeout(()=>{

// bot.message.delete()

// },3000)

message.delete()

message.channel.send("Авторизация прошла успешно ")

if(res[0].id\_r==4){

const embed = new Discord.MessageEmbed()

.setColor('#ff0000')

.setTitle('Преподователь')

.setFooter("!My\_appreciations\n!Mysuggestions\n!My\_visitability\n!All\_users\nGive\_elder")

return message.channel.send(embed);}

if(res[0].id\_r==2){

const embed = new Discord.MessageEmbed()

.setColor('#ff0000')

.setTitle('Староста')

.setFooter("!My\_appreciations\n!Mysuggestions\n!My\_visitability\n!All\_student")

return message.channel.send(embed);

}

if(res[0].id\_r==1){

const embed = new Discord.MessageEmbed()

.setColor('#22467d')

.setTitle('Студент')

.setFooter("!My\_appreciations\n!Mysuggestions\n!My\_visitability")

return message.channel.send(embed);

}

}

else{

message.delete()

return message.channel.send("Неккоректные данные")

}})

};

**A.1.5 Листинги команды Give\_Elder**

/\*

\* Модуль: Give\_Elder

\* Цель: Команда, позволяющая выдать роль старосты студенту

\*/

var con = require('../DB/conDb');

module.exports = async (bot,message,args,argsF) => {

const cont = argsF.join(" ") // все что пишется после! reg

const res1 = cont.split(" ") // Разделяем все записи на массив

var user = message.guild.members.cache.get(message.author.id);

// console.log(user.roles);

// 933467032291266610 преподоавтель

// 931628294409945138 студент

// 257871068532834304 староста

function Check\_User\_id(student,callback){

console.log(student);

// Проверка существует ли Фамилия и Имя студента

con.query('SELECT `id` FROM `users` WHERE `id\_s` = ?',

[student], async (err,res,fields)=>{

//Проверка на выполнение запроса

if(err){

console.log(err.message);

return callback(0,false)

}

if(res.length ==0) {

message.channel.send("такого юзера не существует ")

return callback(0,false) }

else{

console.log(res[0].id)

return callback(res[0].id,true)} })}

function Stage1(student){

Check\_User\_id(student,function (id\_u,res) {

if(res){

con.query('UPDATE `users` SET `id\_r` = 2 WHERE `users`.`id` = ?',

[id\_u], async (err,res,fields)=>{

if(err)

return console.log(err.message);

message.delete()

message.channel.send("Роль у пользователя изменилась")

// var role = message.guild.roles.cache.find(role => role.id === "933467032291266610")

// var user = await bot.users.fetch(id\_u).catch(console.error);

// console.log(user);

const memberID = id\_u; // you want to add/remove roles. Only members have roles not users. So, that's why I named the variable memberID for keeping it clear.

const roleID = '933483006511972352';

const guild = bot.guilds.cache.get('859156583928561705');

const role = guild.roles.cache.get(roleID);

const member = await guild.members.fetch(memberID);

member.roles.add(role);

// var user = message.guild.members.cache.get(message.author.id);

// user.UpdateRole(role.id)

// console.log(id\_u);}) }

else

return console.log("Роль не была обнавлена"); }) }

function Check\_group(callback) {

//Проверка существует ли данная группа

con.query('SELECT \* FROM `groups` WHERE `groups`.`name` =?',

[res1[0]], async (err,res,fields)=>{

//Проверка на выполнение запроса

if(err){

return callback(0,false)}

if(res.length ==0) {

message.channel.send("такой группы не существует")

return callback(0,false) }else{

// берем id группы если такая существует

return callback(res[0].id\_g,true) } })}

function Check\_student(grop,callback) {

// Проверка существует ли Фамилия и Имя студента

con.query('SELECT \* FROM `students` WHERE `students`.`lastname` = ? AND `students`.`firstname` =? AND `students`.`id\_g` = ?',

[res1[1],res1[2],grop], async (err,res,fields)=>{

//Проверка на выполнение запроса

if(err){

console.log(err.message);

return callback(0,false)}

if(res.length ==0) {

console.log(res1[0],res1[1],res1[2]);

message.channel.send("такого студента не существует ")

return callback(0,false)}

else{

return callback(res[0].id\_s,true) } })}

function UpdateRole() {

result = true

// Вызываем метод для проверки на существование группы, если результат true,

// группа найдена, если нет, то выходим и говорим: "Регистрация не была произведена"

//Так же если группа найдена, то в параметр групп записываем id группы

Check\_group(function (grop,result) {

console.log(result);

// Если группа найдена

if(result){

//Тут проблема в том, что параметр !!!!grop!!!! из "Check\_group" я не могу вытащить отсюда

//Я создаю новую переменную, и присваиваю туда grop, но когда выхожу почему то не меняет!!

// Поэтому в этой проверке я вызываю проверку на студента, туда я передаю группу, и в запросе

// смотрю есть ли такой студент или нет

// так же есть параметр student , это id студента, который ищется по фамилии и его имени, и найденному id группы,

// после чего Insert\_reg выполняет запрос на добавление юзера

Check\_student(grop,function (student,result) {

console.log(result);

if(result){

Stage1(student)} else

return console.log("Роль не выдана");}) }

// Если группа не найденаelse

return console.log("Не была выдана роль старосты");})}

if(user.roles.cache.find(role => role.name == "Преподаватель")){

if(res1.length!=3){

message.delete()

return message.channel.send("Для изменения старосты необходимо повторить данную команду !give\_elder `your\_group` `your\_lastname` `your\_firstname`") }

if(res1[0].length>=3 && res1[1].length>3 && res1[2].length>3) UpdateRole() else

user.send("Недопустимая длина данных")}else{

message.delete()

return message.channel.send("Только преподователи могут выдавать старосту")}};

**A.1.6 Листинги команды All\_us**

/\*

\* Модуль: All\_us

\* Цель: Команда, позволяющая просмотреть всех пользователей в системе

\*/

var con = require('../DB/conDb');

module.exports = async (message,Discord) => {

var user = message.guild.members.cache.get(message.author.id) str =""

message.delete()

if(user.roles.cache.find(role => role.name == "Преподаватель")){

con.query('SELECT DISTINCT `students`.`lastname` ,`students`.`firstname` ,`roles`.`Name` ,`groups`.`name` FROM `users`,`students`,`roles`,`groups` WHERE `users`.`id\_s`=`students`.`id\_s` AND `users`.`id\_r`=`roles`.`id\_r` and `groups`.`id\_g`=`students`.`id\_g`',

[], async (err,res,fields)=>{

//Проверка на выполнение запроса

if(err){ console.log(err.message) }

if(!res.length ==0) {

for (let i = 0; i < res.length; i++) {

str+=res[i].lastname+" "+res[i].firstname+" "+res[i].Name+" "+res[i].name+"\n" }

const embed = new Discord.MessageEmbed().setColor('#d8a903')

.setTitle('Все пользователи').setFooter(str)

return user.send(embed) }

else

user.send("Пользователей нет")})} else

user.send("Только преподаватели могут смотреть всех студентов")}

**A.2 Листинги обработки событий в приложении**

**A.2.1 Событие Message**

/\*

\* Модуль: Message

\* Цель: Получить сообщение из чата на сервере, преобразовать в нужный формат и в

\* зависимости от команды, выбрать нужный модуль с данной командой

\*/

module.exports = async (bot, msg,guildMap,connect,Discord) => {

const msgAr = msg.content.toLowerCase().split(' '),

cmd = msgAr[0],

args = msgAr.slice(1),

msgArFull = msg.content.split(' '),

argsF= msgArFull.slice(1)

const user = await bot.users.fetch(msg.author.id)

if(cmd[0]=='!'){

if(cmd=="!auth"){

require('../cmds/aut')(bot,msg,args,argsF,Discord)

return }

if(cmd=="!reg"){

require('../cmds/reg')(bot,msg,args,argsF)

return }

if(cmd=="!give\_elder"){

require('../cmds/give\_elder')(bot,msg,args,argsF)

return

}

if(cmd=="!test"){

require('../cmds/test\_ls')(bot,msg,args,argsF)

return

}

if(cmd=="!join" || cmd =="!leave"){

require('../cmds/join\_leave')(msg,guildMap,connect)

return

}

if(cmd=="!students"){

require('../cmds/students')(msg,Discord)

return

}

if(cmd=="!all\_us"){

require('../cmds/all\_us')(msg,Discord)

return}

user.send("Извините, такой команды не найдено или не существует " + cmd + ".") }}

**A.2.2 Событие Index**

/\*

\* Модуль: Index

\* Цель: Здесь в зависимости от задачи запускается нужная обработка события

\*/

module.exports = (bot,guildMap,connect,Discord) => {

bot

.on('ready', ()=>require('./ready')(bot))

.on('message', (message) => require('./message')(bot, message,guildMap,connect,Discord))

.on('voiceStateUpdate', (oldMember,newMember) => require('./voiceStateUpdate')(oldMember,newMember,guildMap))};

**A.2.3 Событие Ready**

/\*

\* Модуль: Ready

\* Цель: Запускается при старте

\*/

module.exports = async (bot) => {

console.log(`Logged in as ${bot.user.tag}!`)

}

**A.2.4 Событие VoiceStateUpdate**

/\*

\* Модуль: VoiceStateUpdate

\* Цель: Позволяет системе автоматически отключиться, если пользователь вышел

\*/

module.exports = async (oldMember,newMember,guildMap) => {

if(oldMember.channel && !newMember.channel && global.user\_id == newMember.member.user.id ) { //если пользователь вышел из канала

require('../global\_fun/leave')(guildMap,global.Map\_Key\_global,"msg")

} }

**A.2.5 Событие Leave**

/\*

\* Модуль: Leave

\* Цель: Выход из голосового канала, по команде пользователя

\*/

module.exports = async (guildMap,mapKey,msg) => {

if (guildMap.has(mapKey)) {

let val = guildMap.get(mapKey);

if (val.voice\_Channel) val.voice\_Channel.leave()

if (val.voice\_Connection) val.voice\_Connection.disconnect()

guildMap.delete(mapKey)

} else {

msg.reply("Cannot leave because not connected.") }}

**A.3 Главный модуль программы Index**

/\*

\* Модуль: Index

\* Цель: Соединение всех файлов в одном и настройка системы

\*/

var con = require('./DB/conDb');

// LOGGING

function getCurrentDateString() {

return (new Date()).toISOString() + ' ::';

};

\_\_originalLog = console.log;

console.log = function() {

var args = [].slice.call(arguments);

\_\_originalLog.apply(console.log, [getCurrentDateString()].concat(args));

};

const fs = require('fs');

const util = require('util');

const path = require('path');

const { Readable } = require('stream');

// VARIA

function necessary\_dirs() {

if (!fs.existsSync('./data/')) {

fs.mkdirSync('./data/');

}

}

necessary\_dirs()

function sleep(ms) {

return new Promise((resolve) => {

setTimeout(resolve, ms);

});

}

async function convert\_audio(input) {

try {

// stereo to mono channel

const data = new Int16Array(input)

const ndata = data.filter((el, idx) => idx % 2);

return Buffer.from(ndata);

} catch (e) {

console.log(e)

console.log('convert\_audio: ' + e)

throw e; }}

//CONFIG

const SETTINGS\_FILE = 'settings.json';

// let DISCORD\_TOK = 'OTM2NzEzNDM2NDU5NTkzNzc4.YfRMXg.M1IWZumAxfEugqwQFg2j1PoXa4w';

// let WITAI\_TOK = null;

// let SPEECH\_METHOD = 'vosk'; // witai, google, vosk

function loadConfig() {

if (fs.existsSync(SETTINGS\_FILE)) {

const CFG\_DATA = JSON.parse(fs.readFileSync(SETTINGS\_FILE, 'utf8'));

DISCORD\_TOK = CFG\_DATA.DISCORD\_TOK;

WITAI\_TOK = CFG\_DATA.WITAI\_TOK;

SPEECH\_METHOD = CFG\_DATA.SPEECH\_METHOD; }

DISCORD\_TOK = process.env.DISCORD\_TOK || DISCORD\_TOK;

WITAI\_TOK = process.env.WITAI\_TOK || WITAI\_TOK;

SPEECH\_METHOD = process.env.SPEECH\_METHOD || SPEECH\_METHOD;

if (!['witai', 'google', 'vosk'].includes(SPEECH\_METHOD))

throw 'invalid or missing SPEECH\_METHOD'

if (!DISCORD\_TOK)

throw 'invalid or missing DISCORD\_TOK'

if (SPEECH\_METHOD === 'witai' && !WITAI\_TOK)

throw 'invalid or missing WITAI\_TOK'

if (SPEECH\_METHOD === 'google' && !fs.existsSync('./gspeech\_key.json'))

throw 'missing gspeech\_key.json'}

loadConfig()

const Discord = require('discord.js')

const DISCORD\_MSG\_LIMIT = 2000;

const discordClient = new Discord.Client()

discordClient.login(DISCORD\_TOK)

const guildMap = new Map();

require('./events')(discordClient, guildMap, connect, Discord)

function Dictionary() {

this.dataUser = []

this.add = function(key, value) {

if (key && value) {

this.dataUser.push({

key: key,

value: value

})

return this.dataUser

}

}

this.findAt = function(key) {

for (var i = 0; i < this.dataUser.length; i++) {

if (this.dataUser[i].key === key) {

return this.dataUser[i].value

}

}

return "Not found "

}

this.size = function() {

return this.dataUser.length

}

}

var dict = new Dictionary()

function Select\_dict(dict, callback) {

// Проверка существует ли Фамилия и Имя студента

con.query('SELECT `users`.`id` , CONCAT(`students`.`lastname`," ",`students`.`firstname`) AS "FN" FROM `users`,`students` WHERE `users`.`id\_s`=`students`.`id\_s`', [], async(err, res, fields) => {

//Проверка на выполнение запроса

if (err) {

console.log(err.message);

return callback(0) }

if (res.length != 0) {

for (let i = 0; i < res.length; i++) {

dict.add(res[i].id, res[i].FN) }

return callback(dict) }})}

const SILENCE\_FRAME = Buffer.from([0xF8, 0xFF, 0xFE]);

class Silence extends Readable {

\_read() {

this.push(SILENCE\_FRAME);

this.destroy(); }}

async function connect(msg, mapKey, id\_r) {

try {

let voice\_Channel = await discordClient.channels.fetch(msg.member.voice.channelID);

if (!voice\_Channel) return msg.reply("Error: The voice channel does not exist!");

let text\_Channel = await discordClient.channels.fetch(msg.channel.id);

if (!text\_Channel) return msg.reply("Error: The text channel does not exist!");

let voice\_Connection = await voice\_Channel.join();

voice\_Connection.play(new Silence(), { type: 'opus' });

guildMap.set(mapKey, {

'text\_Channel': text\_Channel,

'voice\_Channel': voice\_Channel,

'voice\_Connection': voice\_Connection,

'selected\_lang': 'ru',

'debug': false, });

speak\_impl(voice\_Connection, mapKey, id\_r)

voice\_Connection.on('disconnect', async(e) => {

if (e) console.log(e);

guildMap.delete(mapKey);

})

msg.reply('connected!')

} catch (e) {

console.log('connect: ' + e)

msg.reply('Error: unable to join your voice channel.');

throw e; }}

const vosk = require('vosk');

let recs = {}

if (SPEECH\_METHOD === 'vosk') {

vosk.setLogLevel(-1);

// MODELS: https://alphacephei.com/vosk/models

recs = {

'ru': new vosk.Recognizer({ model: new vosk.Model('vosk\_models/ru'), sampleRate: 48000 }),

// 'fr': new vosk.Recognizer({model: new vosk.Model('vosk\_models/fr'), sampleRate: 48000}),

// 'es': new vosk.Recognizer({model: new vosk.Model('vosk\_models/es'), sampleRate: 48000}), }

// download new models if you need

// dev reference: https://github.com/alphacep/vosk-api/blob/master/nodejs/index.js}

function speak\_impl(voice\_Connection, mapKey, id\_r) {

voice\_Connection.on('speaking', async(user, speaking) => {

if (speaking.bitfield == 0 || user.bot) { return }

console.log(`I'm listening to ${user.username}`)

// this creates a 16-bit signed PCM, stereo 48KHz stream

const audioStream = voice\_Connection.receiver.createStream(user, { mode: 'pcm' })

audioStream.on('error', (e) => {

console.log('audioStream: ' + e)});

let buffer = [];

audioStream.on('data', (data) => {

buffer.push(data)})

audioStream.on('end', async() => {

buffer = Buffer.concat(buffer)

const duration = buffer.length / 48000 / 4;

console.log("duration: " + duration)

try {

let new\_buffer = await convert\_audio(buffer)

let out = await transcribe(new\_buffer, mapKey);

if (out != null) {

process\_commands\_query(out, mapKey, user);

Select\_dict(dict, function(dict) {

if (out == "")

con.query('CALL `add\_rec\_dur`(?, ?)', [duration, id\_r], async(err,fields=> {

if (err)

return console.log(err.message);})

else

con.query('CALL `add\_rec`(?, ?, ?, ?)', [dict.findAt(user.id), id\_r, duration, out], async(err, fields) => {

if (err)

return console.log(err.message);})})

con.query('CALL `add\_speach`(?, ?)', [user.id, out], async(err, fields) => {

if (err)

return console.log(err.message); }) }

} catch (e) {

console.log('tmpraw rename: ' + e) } }) })}

function process\_commands\_query(txt, mapKey, user) {

if (txt && txt.length) {

let val = guildMap.get(mapKey);

val.text\_Channel.send(user.username + ': ' + txt) }}

// SPEECH

async function transcribe(buffer, mapKey) {

if (SPEECH\_METHOD === 'witai') {

return transcribe\_witai(buffer)

} else if (SPEECH\_METHOD === 'google') {

return transcribe\_gspeech(buffer)

} else if (SPEECH\_METHOD === 'vosk') {

let val = guildMap.get(mapKey);

recs[val.selected\_lang].acceptWaveform(buffer);

let ret = recs[val.selected\_lang].result().text;

console.log('vosk:', ret)

return ret; }}

async function transcribe\_gspeech(buffer) {

try {

console.log('transcribe\_gspeech')

const bytes = buffer.toString('base64');

const audio = {

content: bytes, };

const config = {

encoding: 'LINEAR16',

sampleRateHertz: 48000,

languageCode: 'en-US', // https://cloud.google.com/speech-to-text/docs/languages};

const request = {audio: audio, config: config, };

const [response] = await gspeechclient.recognize(request);

const transcription = response.results

.map(result => result.alternatives[0].transcript) .join('\n');

console.log(`gspeech: ${transcription}`);

return transcription; } catch (e) { console.log('transcribe\_gspeech 368:' + e) }

**A.4 Листинг класса ConDb**

/\*

\* Класс: ConDb

\* Цель: Подключение к базе данных

\*/

const mysql = require("mysql");

process.argv[1]= 0

const connection = mysql.createConnection({

host: "localhost", port: "3306",

user: "root", database: "bot\_discrod", password: "root" });

connection.connect(function(err){

if (err) {

return console.error("Ошибка: " + err.message); }

else{

console.log("Подключение к серверу MySQL успешно установлено");}});

module.exports = connection

ПРИЛОЖЕНИЕ Б

(обязательное)

Результаты выполнения программы

На рисунке Б.1 представлено окно консоли сервера приложений.

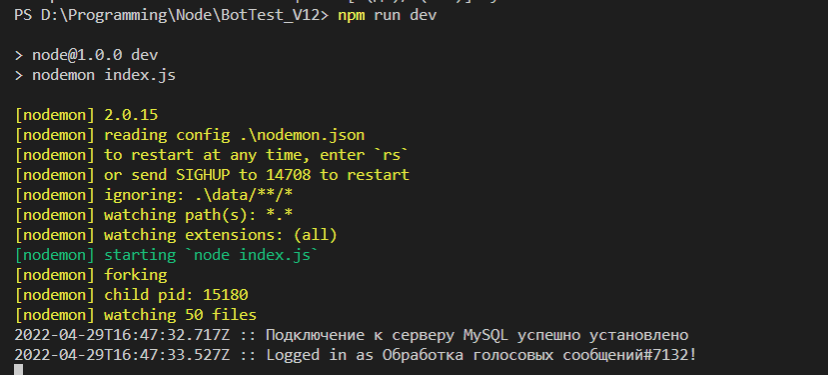


Рисунок Б.1 – Окно консоли сервера приложений

На рисунке Б.2 представлена команда входа в систему.

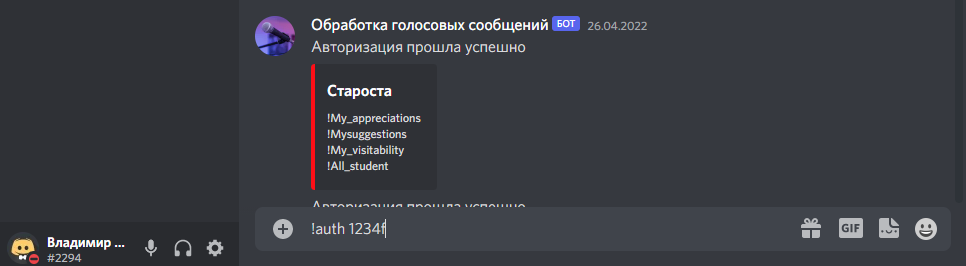


Рисунок Б.2 – Команда входа в систему

На рисунке Б.3 представлена команда регистрации в системе.

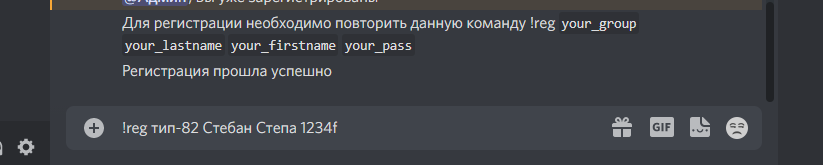


Рисунок Б.3 – Команда регистрации в системе

На рисунке Б.4 представлена команда для получение всех пользователей.

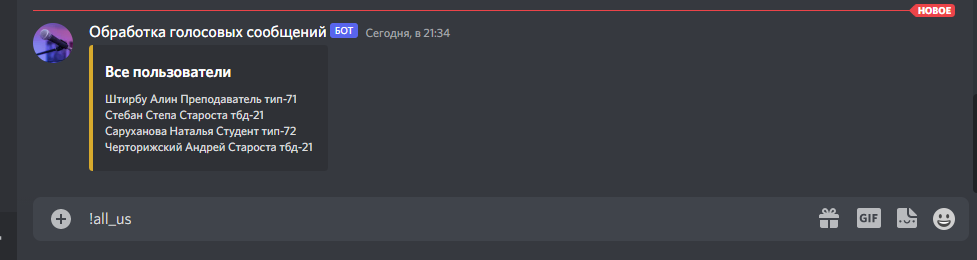


Рисунок Б.4 – Команда получение всех пользователей

На рисунке Б.5 представлена команда просмотра студентов группы.

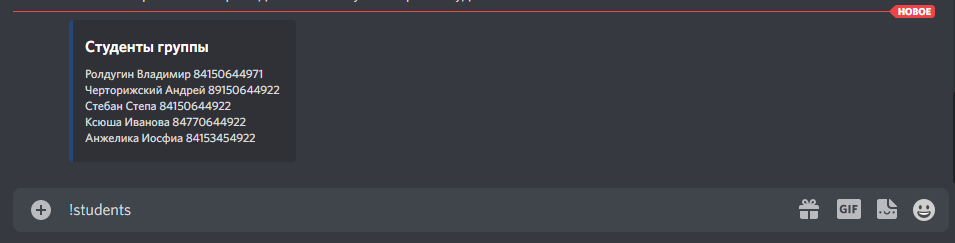


Рисунок Б.5 – Команда просмотра студентов группы

На рисунке Б.6 представлена команда подключения системы.

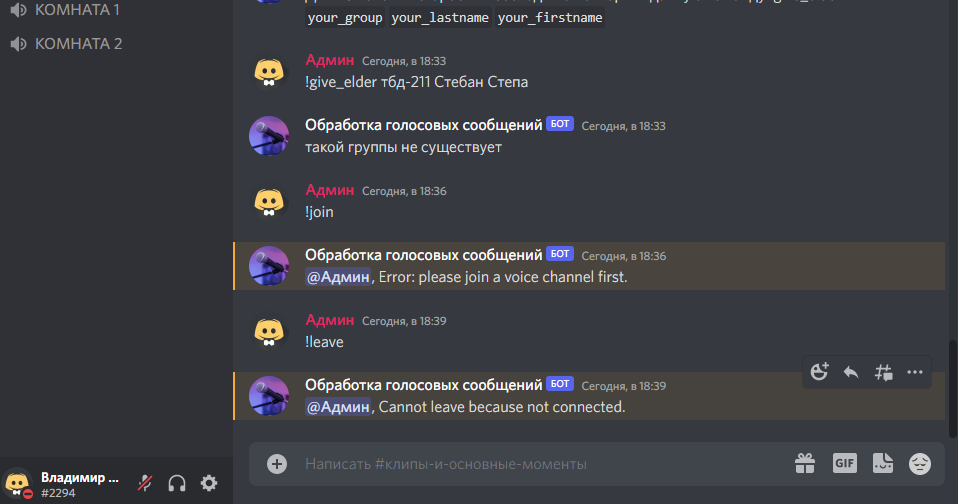


Рисунок Б.6 – Команда подключения системы

На рисунке Б.7 представлена команда отключения системы.

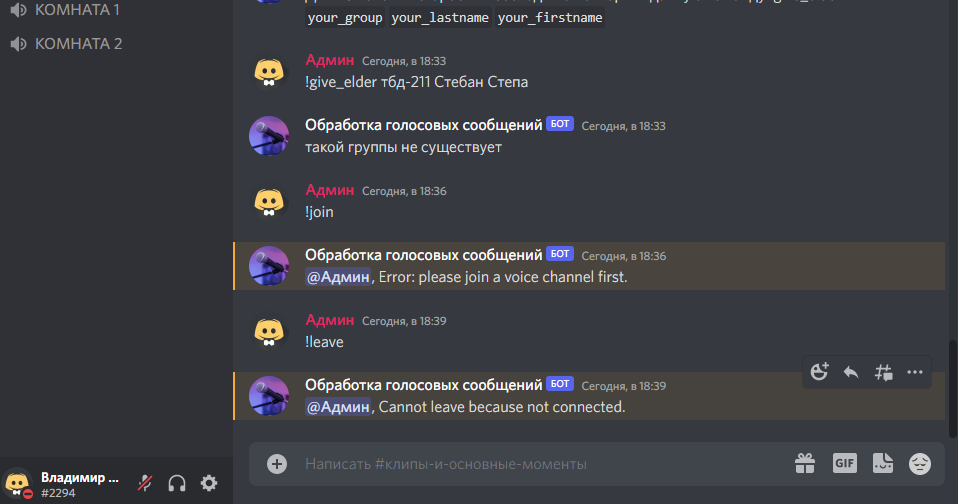


Рисунок Б.7 – Команда отключения системы

На рисунке Б.8 представлена команда добавления роли.



Рисунок Б.8 – Команда добавления роли