МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Теоретической и прикладной информатики
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Чубич В.М.
(фамилия, имя, отчество)

(подпись)

«_____»_____ 2021 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Сычева Егора Леонидовича
(фамилия, имя, отчество студента – автора работы)

Разработка чат-бота на основе различных моделей обработки естественного языка
(тема работы)

Факультет Прикладной математики и информатики
(полное название факультета)

Направление подготовки 02.03.03. Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки бакалавра)

Руководитель от НГТУ	**Автор выпускной квалификационной работы**
Аврунев О.Е. (фамилия, имя, отчество)	Сычев Е.Л. (фамилия, И.О.)
(ученая степень, ученое звание)	ФПМИ, ПМИ-71 (факультет, группа)
(подпись, дата)	(подпись, дата)

Новосибирск, 2021 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Теоретической и прикладной информатики
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Чубич В.М.
(фамилия, имя, отчество)

«9» марта 2021 г

(подпись)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

студенту ***Сычеву Егору Леонидовичу***
(фамилия, имя, отчество студента)

Направление подготовки *02.03.03. Математическое обеспечение и администрирование информационных систем*

Факультет Прикладной математики и информатики

Тема ***Разработка чат-бота на основе различных моделей обработки естественного языка***

Исходные данные (или цель работы):

Разработка чат-бота на платформе мессенджера Telegram с распознаванием вопросов на русском языке и ответа на них

Структурные части работы:

Работа состоит из трёх частей:

1. *В разделе "Анализ и проектирование" описывается портрет целевой аудитории и разрабатывается структура приложения*
2. *В разделе "Выбор инструментов реализации" рассматриваются различные языки программирования и библиотеки, выбираются оптимальные для реализации поставленной задачи*
3. *В разделе "Описание программы" представлена детальная информация о программе и требованиях для её функционирования*

Задание согласовано и принято к исполнению.

Руководитель от НГТУ	**Студент**
Аврунев О.Е.	*Сычев Е.Л.*
(фамилия, имя, отчество)	(фамилия, имя, отчество)
	ФПМИ, ПМИ-71
(ученая степень, ученое звание)	(факультет, группа)
09.03.2021 г.	*09.03.2021 г.*
(подпись, дата)	(подпись, дата)

Тема утверждена приказом по НГТУ № *969/2* от «*9*» *марта 2021* г.

ВКР сдана в ГЭК № _____, тема сверена с данными приказа

9 марта 2021 г.
(подпись секретаря экзаменационной комиссии по защите ВКР, дата)

Волкова В.М.
(фамилия, имя, отчество секретаря экзаменационной комиссии по защите ВКР)

АННОТАЦИЯ

Отчет 51 с., 3 ч., 4 рис., 11 источников, 2 прил.

РАЗРАБОТЧКА ЧАТ-БОТА НА ОСНОВЕ РАЗЛИЧНЫХ МОДЕЛЕЙ ОБРАБОТКИ ЕСТЕСТВЕННОГО ЯЗЫКА.

Объектом разработки является чат-бот на платформе Telegram, способный распознавать вопросы на русском языке и давать на них ответ в рамках предметной области.

Цель работы – реализация функционирующего чат-бота на платформе мессенджера Telegram с пониманием текстов на русском языке.

В процессе работы проводились сравнения библиотек для распознавания естественных языков, спроектирована и реализована база данных для поддержки чат-бота.

В результате был создан чат-бот на платформе мессенджера Telegram, понимающий вопросы на русском языке.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Каждый год НГТУ проводит приёмную кампанию и обрабатывает до нескольких тысяч заявок от абитуриентов. Для этого работает приёмная комиссия, в обязанности которой входит, в том числе, и давать ответы на вопросы поступающих. Для упрощения работы комиссии на сайте университета есть вся необходимая информация и раздел с часто задаваемыми вопросами и ответами. Тем не менее, абитуриенты не всегда могут найти нужную информацию, поэтому задают вопросы непосредственно приёмной комиссии, а так как её время работы имеет определённое расписание, то ответ может быть получен не скоро. Для решения этой проблемы и для разгрузки приёмной комиссии, было решено создать чат-бот для мессенджера Telegram.

Чат-бот – это программа автоматического общения с пользователем посредством обмена текстовыми или голосовыми сообщениями. Для этого бот должен понимать намерения пользователя – интенты.

Распознавание интентов необходимо для выбора ответа, который ожидает получить пользователь и попадает в контекст разговора. Эта задача является AI-полной, то есть задачей, которую легко решает человек, но которую сложно решить алгоритмически. Для ответа пользователю, как правило, используются заранее заготовленные варианты ответа.

В этой работе описываются процессы решения описанных выше задач. В первой главе проводится анализ целевой аудитории, предметной области и проектируется архитектура приложения.

Во второй главе выбираются инструменты реализации программы: язык программирования, библиотек обработки текстов на естественном языке и Telegram API. Также в этой главе выбираются инструменты для создания веб-сервера и взаимодействия с базой данных.

В третьей главе представлено описание итоговой программы, необходимые инструменты для её запуска и интерфейса взаимодействия с базой данных для редактирования бота.

1 АНАЛИЗ И ПРОЕКТИРОВАНИЕ

1.1 Анализ целевой аудитории

Для понимания необходимости чат-бота и выбора платформы необходимо учитывать специфику аудитории, для которой будет создаваться чат-бот. Наша аудитория – молодые люди возрастом до 25 лет, владеющие русским языком и которые выбирают или уже поступают в НГТУ после школы или среднего специального (профессионального) учебного заведения.

Наиболее распространёнными и привычными каналами связи для таких людей будут социальные сети и мессенджеры. Наиболее популярными являются “ВКонтакте”, “Instagram”, “WhatsApp” и “Telegram”. На текущий момент “WhatsApp” не имеет открытых инструментов для создания чат-бота, а “Instagram” в большей степени ориентирован на публикацию и просмотр фото- и видеоконтента. Также в последнее время наблюдается тенденция перехода молодых людей из “ВКонтакте” в “Telegram”, поэтому последний является более предпочтительной площадкой для создания чат-бота.

1.2 Анализ предметной области

Чтобы понять необходимость создания чат-бота, рассмотрим несколько сценариев, в которых поступающий ищет ответы на свои вопросы по поводу поступления (см. Рисунок 1 – Диаграмма прецедентов для поступающего).

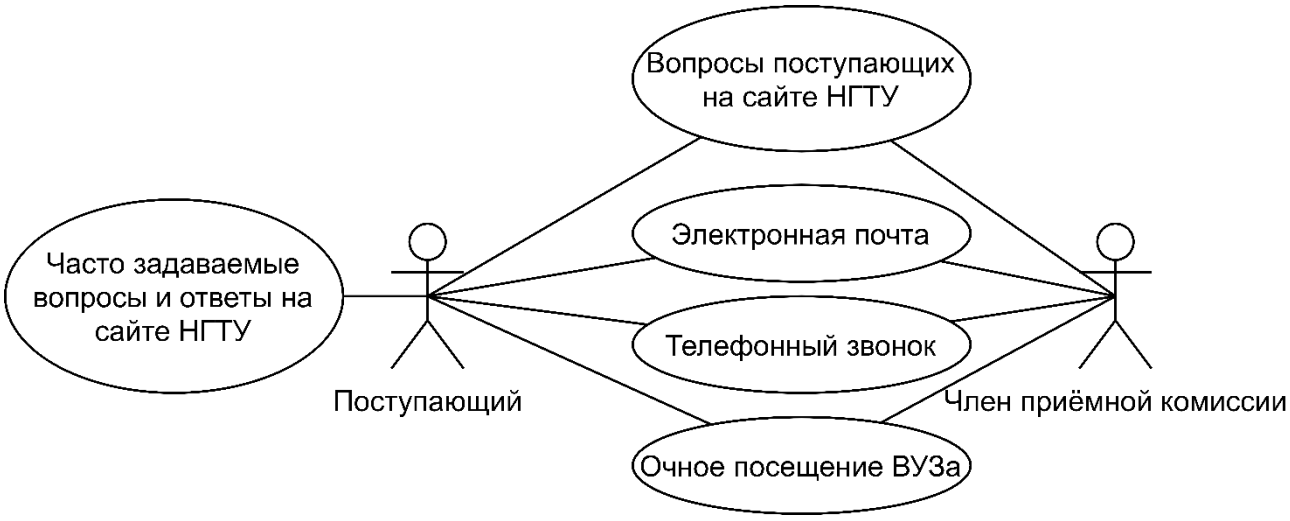

Рисунок 1 – Диаграмма прецедентов для поступающего

У данной системы есть два недостатка:

1) не удобно регулярно обращаться к часто задаваемым вопросам и ответам, особенно с мобильных устройств;
2) ответ от приёмной комиссии, скорее всего, придётся с задержкой, от нескольких минут до нескольких дней, в зависимости от её загруженности.

В такой ситуации использование чат-бота способно обеспечить более простой доступ к часто задаваемым вопросам и ответам. Также, при необходимости, можно адресовать вопрос напрямую приёмной комиссии и получить в диалоге с чат-ботом.

Использование технологий обработки естественного языка позволит поступающему не искать свой вопрос в списке часто задаваемых, а сформулировать текстом и сразу получить ответ.

1.3 Проектирование архитектуры приложения

Приложение должно работать на платформе мессенджера “Telegram”, уметь распознавать текстовые вопросы на русском языке в контексте поступления в НГТУ и давать на них ответы.

Должна быть реализована база данных для хранения вопросов и ответов на них. База данных необходима для возможности дополнительного обучения и совершенствования чат-бота, расширения списка возможных ответов. Для удобства поддержания приложения, необходимо организовать аутентифицированный доступ к базе данных.

Для упрощения переносимости приложения между различными хост-машинами, приложение будет реализовано в виде микросервисов, каждому из которых будет отведён свой контейнер. Такая архитектура позволит не настраивать каждый раз окружение.

Схема взаимодействия различных модулей и пользователей представлена ниже Рисунок 2 – Схема взаимодействия программных модулей.

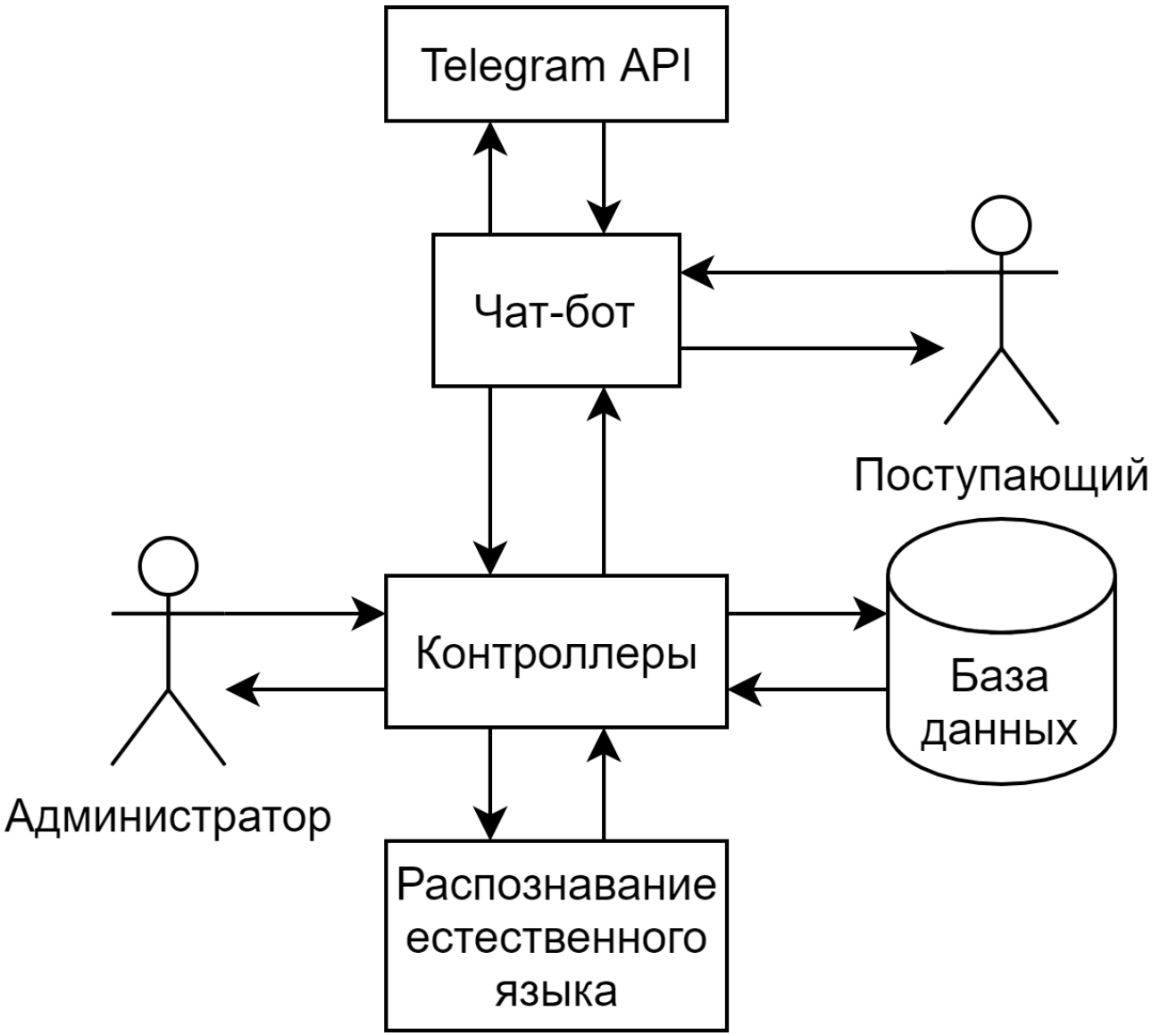

Рисунок 2 – Схема взаимодействия программных модулей

Диаграмма прецедентов приложения представлена ниже на Рисунок 3 – Диаграмма прецедентов приложения.

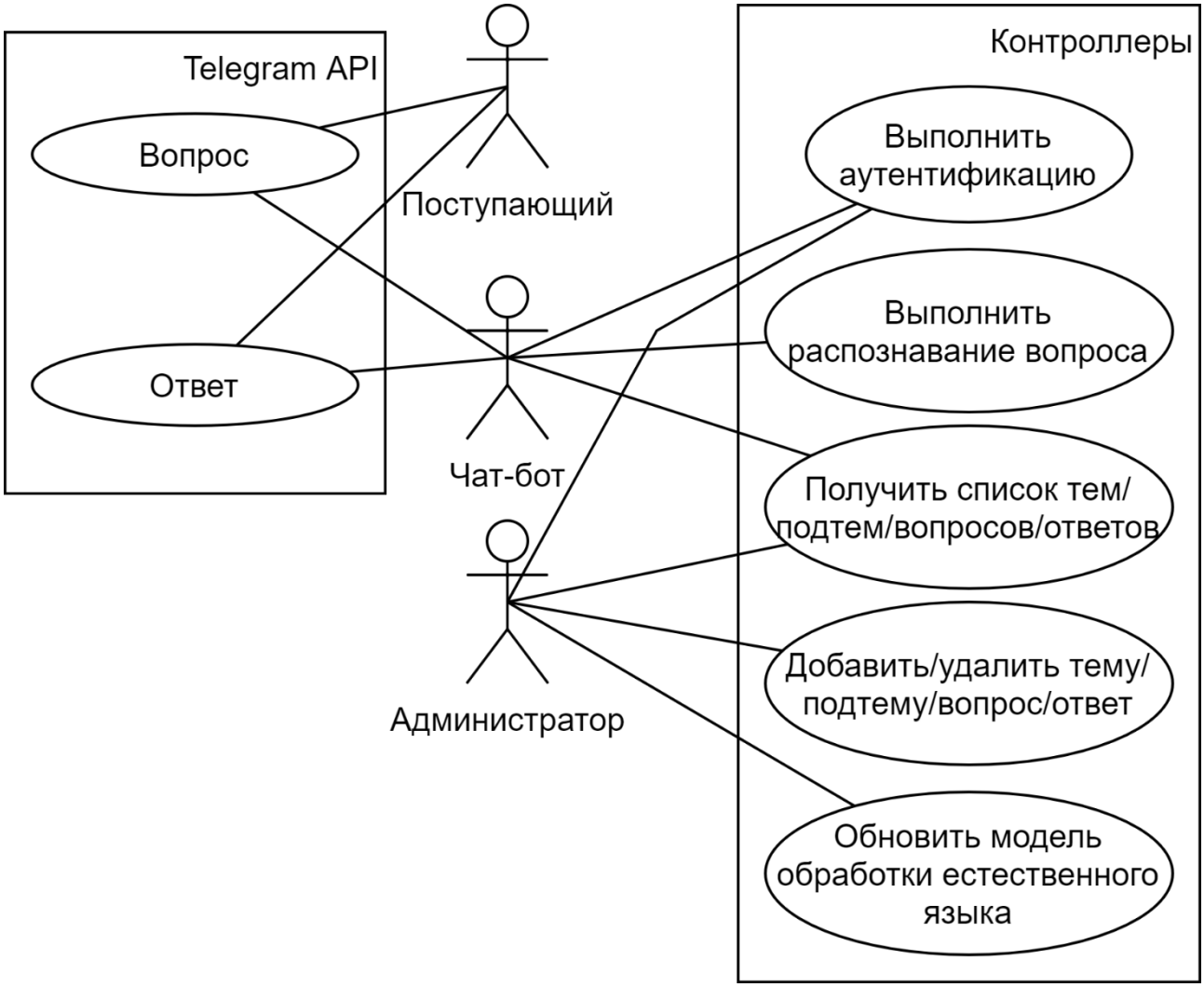

Рисунок 3 – Диаграмма прецедентов приложения

Структура базы данных представлена ниже на Рисунок 4 – Структура базы данных.

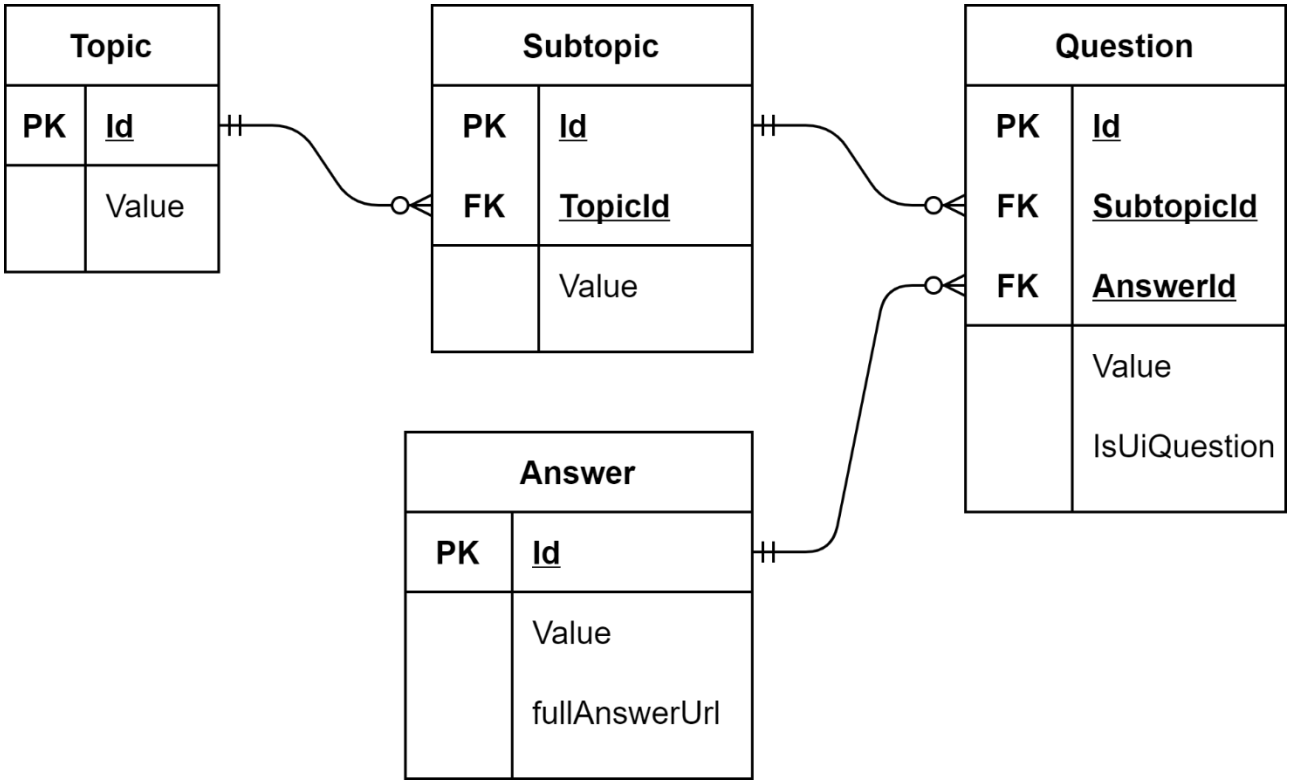

Рисунок 4 – Структура базы данных

2 ВЫБОР ИНСТРУМЕНТОВ РЕАЛИЗАЦИИ

2.1 Выбор языков программирования

Для создания приложения необходимо реализовать три компонента:

1) сервер контроллеров;
2) модуль обработки текстов на естественном языке;
3) непосредственно чат-бот.

Для создания сервера контроллеров проще всего использовать язык программирования C# – компилируемый язык высокого уровня, объектно-ориентированный, с сильной статической типизацией. Этот язык наиболее предпочтителен, так как для него уже есть готовые фреймворки, реализующие весь необходимый функционал. К тому же, C# интенсивно набирает популярность в качестве языка программирования именно для серверной части приложений.

Для обработки текстов на естественном языке лучше подойдёт язык программирования Python – интерпретируемый язык высокого уровня с сильной динамической типизацией. Python позволяет писать очень короткий и читаемый код, но он не годится для реализации высоконагруженных систем. Тем не менее, это компенсируется библиотеками, написанными на более производительных языках программирования. Таким образом, Python позволяет комбинировать лаконичный и читаемый код с довольно высокой производительностью.

Для реализации непосредственно чат-бота можно использовать Python по тем же причинам, что описаны выше.

2.2 Выбор системы управления базой данных

Для хранения данных можно использовать реляционную или не реляционную систему управления базой данных. Сравнение этих систем представлено ниже в Таблица 1 – Сравнение систем управления базой данных.

Таблица 1 – Сравнение систем управления базой данных

Реляционная СУБД	Не реляционная СУБД
Необходимость в предопределённой неизменяемой структуре данных.	Отсутствует необходимость в предопределённой структуре данных. Схема может меняться.
Ограничения данных на уровне СУБД.	Данные никак не проверяются.
Данные всегда согласованы.	В определённый момент времени данные могут быть не согласованы.
Вертикальная масштабируемость.	Горизонтальная масштабируемость.
Оптимизация запросов.	Запросы не оптимизируются.

Так как схема данных достаточно проста и не изменяема, то более предпочтительным вариантом будет выбор реляционной системы управления базой данных.

Среди бесплатных реляционных СУБД наиболее популярными являются PostgreSQL, MySQL и SQLite. SQLite не использует серверную часть, а вся база данных хранится в одном файле на том же компьютере, на котором запускается клиент, её использующий. MySQL частично не соответствует SQL и уступает PostgreSQL по эффективности. Таким образом, наиболее предпочтительной СУБД является PostgreSQL.

2.3 Выбор библиотеки для взаимодействия с СУБД

Прямой доступ к базе данных по средством SQL-запросов не удобен, потому что необходимо следить за правильностью написания текста запроса и держать в голове структуру базы данных. Эту проблему решает ADO.NET Entity Framework для языка C#. Этот фреймворк является объектно-ориентированной технологией доступа к данным и позволяет прямо в коде программы писать запросы к базе данных с учётом типов данных и проверкой син-

таксиса. Для создания запросов используется синтаксис LINQ (Language-Integrated Query) или использовать аналогичные функции высшего порядка.

2.4 Выбор библиотеки для реализации контроллеров

Для реализации контроллеров в сочетании с Entity Framework проще всего использовать фреймворк ASP.NET Core. Этот фреймворк позволяет писать веб-приложения в архитектуре модель-представление-контроллер, хотя в нашем случаем компонент “представление” избыточен и не будет реализован. Такое сочетание фреймворков позволит один раз описать сущности в базе данных (модели) и использовать их и для работы с базой данных, и для работы с чат-ботом.

Вместе с ASP.NET Core прилагается веб-сервер, поэтому нет необходимости в его создании и настройки как в отдельном компоненте. Также ASP.NET Core позволяет легко реализовать аутентификацию, а также входные и выходные данные и маршрутизацию для контроллеров.

2.5 Выбор библиотеки обработки текстов

Для решения задачи понимания естественного языка есть такие библиотеки как NLTK для обработки естественного языка и scikit-learn для решения задачи машинного обучения и построения моделей классификации намерений и выделения сущностей из текстов. Также есть библиотеки для работы с нейронными сетями: TensorFlow, Keras и PyTorch, которые тоже решают задачу машинного обучения. Проблема этих библиотек в том, что они сравнительно низкоуровневые и при их использовании придётся самостоятельно строить модель обучения и конвейер обработки пользовательских запросов.

Вместо библиотек, предназначенных для непосредственно машинного обучения, можно использовать более высокоуровневые библиотеки, которые позволяют написать простую конфигурацию для решения указанных выше проблем. К таким библиотекам относятся DeepPavlov и Rasa. Обе библиотеки являются бесплатными и имеют открытый исходный код, но при этом Rasa го-

раздо дольше развивается, его сообщество больше и с его помощью проще настраивать модель обработки естественного языка.

Для улучшения качества распознавания, можно использовать машинную коррекцию опечаток. В целом эта задача является довольно сложно и вычислительно затратной, тем не менее, есть довольно эффективные библиотеки, которые дают хорошее качество распознавания и работают достаточно быстро. Одна из таких библиотек – JamSpell.

3 ОПИСАНИЕ ПРОГРАММЫ

3.1 Общие сведения

3.1.1 Необходимое программное обеспечение

1) любая из следующих операционных систем: Microsoft Windows, FreeBSD, Linux, macOS;
2) программа Docker и утилита Docker Compose;
3) программа cURL или другая программа, позволяющая осуществлять запросы к веб-серверу по протоколу HTTP;
4) клиент Telegram;

3.1.2 Языки программирования, на которых написана программа

Чат-бот написан на языке Python, веб-сервер реализован на языке C#, а модуль распознавания текстов на естественном языке написан в парадигме low-code e, то есть вместо использования библиотечных функций на каком-то языке программирования используются конфигурационные файлы для используемой библиотеки Rasa, написанные на языке сериализации данных YAML. Тем не менее, для распознавания написаны два дополнительных компонента на языке Python.

3.2 Функциональное назначение

Программа предоставляет заранее заданные ответы на вопросы, написанные на русском языке, посредством общения через мессенджер “Telegram”.

3.3 Описание логической структуры

3.3.1 Алгоритм программы

1) создание докер-образов СУБД PostgreSQL, модуля распознавания, веб-сервера и чат-бота;

2) запуск всех контейнеров утилитой Docker Compose;
3) регистрация первого пользователя в системе;
4) аутентификация первого пользователя;
5) регистрация дополнительных пользователей;
6) аутентификация пользователя;
7) добавление вопросов и ответов посредство HTTP-запросов;
8) обучение модели;
9) ввод пользователя через Telegram;
10) распознавание смысла и выбор ответа;
11) ответ пользователю через Telegram.

3.3.2 Используемые методы

Для поддержания платформенной независимости программы используется технология контейнеризации, то есть по платформенно независимому описанию создаётся образ операционной системы, содержащий саму программу и все необходимые зависимости.

Для реализации веб-сервера используется фреймворк ASP.NET Core, а для взаимодействия с базой данных используется Entity Framework.

3.3.3 Структура программы с описанием составных частей

1) Файлы конфигурации для программы Docker: Dockerfile и docker-compose.yml, необходимые для правильной контейнеризации всей программы. Это нужно для поддержания платформенной независимости приложения.
2) Файлы модуля распознавания естественного языка:
    - text_preprocessing.py содержит реализацию компонента предобработки сообщений;
    - spelling_correction.py содержит реализацию компонента исправления опечаток.
3) Файлы веб-сервера:

- Database.csproj содержит описание проекта и используемых библиотек;
- appsetings.json содержит конфигурацию приложения;
- Properties/launchSettings.Json содержит параметры запуска приложения;
- nlu_config.yml содержит конфигурацию пайплайна для модуля распознавания естественного языка;
- Controllers/ в этой папке содержаться реализации контроллеров для всех сущностей, а также для аутентификации и сервиса NLU.
- Data/QAContext.cs содержит описание схемы базы данных;
- Domain/AuthRequest.cs содержит описание тела запроса для аутентификации;
- Domain/AuthResult.cs содержит описание тела ответа аутентификации;
- Domain/AddAnswerRequest.cs содержит описание тела запроса для добавления ответа в базу данных;
- Domain/ParseRequest.cs содержит описание тела запроса для определения ответа по тексту вопроса;
- Models/ содержит описание сущностей базы данных;
- Nlu/ содержит описание сущностей, возвращаемых сервисом NLU;
- Program.cs содержит точку входа в пространство и запускает веб-сервер;
- Startup.cs выполняет конфигурацию веб-сервера.

3.3.4 Связи программы с другими программами

Программа взаимодействует с клиентом Telegram пользователя посредством Telegram API через программы чат-бота. Чат-бот взаимодействует с веб-сервером посредством HTTP-запросов. Также веб-сервер служит связующим программным обеспечением для сервиса распознавания.

3.4 Используемые технические средства

Чат-бот расположен на сервере облачной платформы НГТУ со следующей конфигурацией:

1) операционная система: Ubuntu Server 20.04;
2) процессор: Intel Core, 4 ядра;
3) оперативная память: 2 ГБ;
4) твердотельный накопитель: 20 ГБ.

3.5 Вызов и загрузка

Запуск программы осуществляется утилитой Docker Compose командой docker-compose up –build --detach.

3.6 Входные данные

Предварительных данных для запуска программы не требуется. Во время работы программа ожидает текстовые сообщения в мессенджере Telegram или HTTP-запросы. Полный список запросов и их параметров приведён в ПРИЛОЖЕНИЕ А.

3.7 Выходные данные

По завершении работы программа не создаёт каких-то выходных данных. Во время работы программа только отвечает на текстовые сообщения в мессенджере Telegram и на HTTP-запросы. Полный список запросов и их ответов приведён в ПРИЛОЖЕНИЕ А.

ЗАКЛЮЧЕНИЕ

Целью данной работы было создание дополнительного информационного канала для поступающих, через который они могли бы узнать необходимую информацию о поступлении в НГТУ.

В ходе выполнения работы было проведено исследование целевой аудитории и предметной области, обоснована необходимость создание чат-бота. Спроектирован и реализован чат-бот на платформе мессенджера Telegram, использующий технологию распознавания текстов на естественном языке. Проведено сравнение двух фреймворков для распознавания текста: Rasa и DeepPavlov, а также нескольких систем управления базами данных.

Проект можно развивать и дальше: добавить административную панель для просмотра статистика использования чат-бота и более удобным взаимодействием с базой данных, улучшать данные для обучения распознавания и проверки правописания.

Тем не менее, цель можно считать достигнутой, так как получившийся чат-бот отвечает всем функциональным требованиям, а его модель распознавания достаточно легко можно улучшать. Для увеличения производительности можно запустить бота на нескольких компьютерах и распределить нагрузку между ними.

СПИСОК ЛИТЕРАТУРЫ

1. Охеда Тони Прикладной анализ текстовых данных на Python. Машинное обучение и создание приложений обработки естественного языка / Охеда Тони, Бенгфорт Бенджамин, Билбро Ребекка – СПБ.: Питер, 2019. – 368 с.: ил.
2. Данарсанам Срини Разработка чат-ботов и разговорных интерфейсов / пер. с англ. М. А. Райтман – М.: ДМК Пресс, 2019. – 340 с.
3. Остроух А.В. Проектирование информационных систем / А.В. Остроух, Н.Е. Суркова – СПБ.: Лань, 2019. – 164 с.
4. Хобсон Лейн Коул Ховард, Ханнес Хапке Обработка естественного языка в действии / пер. с англ. И. Пальти, С. В. Черников – СПБ.: Питер, 2020. – 576 с.
5. Microsoft Документация по ASP.NET [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/aspnet/core/?view=aspnetcore-5.0, свободный – 20.05.2021
6. Microsoft Документация по Entity Framework [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/ef/, свободный – 20.05.2021
7. Ela Kumar Natural Language Processing. – New Delhi : I. K. International Publishing House Pvt. Ltd., 2011. – 224 с.
8. Mike Amundsen RESTful Web APIs / Mike Amundsen, Sam Ruby, Leonard Richardson – [s.l.] : O'Reilly Media, Inc., 2013. – 406 с.
9. DeepPavlov documentation [Электронный ресурс]. – Режим доступа: http://docs.deeppavlov.ai/en/master/index.html, свободный – 05.05.2021
10. Raj Sumit Building Chatbots with Python. – [s.l.]: Apress, 2018. – 224 с.
11. Rasa documentation [Электронный ресурс]. – Режим доступа: https://rasa.com/docs/rasa/, свободный – 05.05.2021

ПРИЛОЖЕНИЕ А. ОПИСАНИЕ HTTP-ЗАПРОСОВ

- GET /AuthManagement/RegisterFirst регистрация первого пользователя. Возвращает статус 200, если регистрация прошла успешно, 400 если email уже зарегистрирован или 500 с текстом ошибки в теле ответа.
- POST /AuthManagement/Register регистрация пользователя. Ожидает в теле запроса JSON-объект со строковыми полями email и password. Возвращает статус 200 и JSON-объект с текстовым полем Token и целочисленным полем Expire, если регистрация прошла успешно, 400 если email уже зарегистрирован или 500 с текстом ошибки в теле ответа.
- POST /AuthManagement/Login аутентификация. Ожидает в теле запроса JSON-объект со строковыми полями email и password. Возвращает 200 и JSON-объект с текстовым полем Token и целочисленным полем Expire, если аутентификация прошла успешно, 404 если пользователь не найден или 400, в случае других ошибок.
- GET /{Topics|Subtopics|Questions|Answers}/get/all получить список сущностей. Ожидает целочисленные параметры в запросе offset и size. Возвращает статус 200 и список сущностей в интервале [offset; offset + size).
- GET /{Topics|Subtopics|Questions|Answers}/get/{id}/ получить сущность по её идентификатору. Ожидает целочисленный параметр id. Возвращает статус 200 и сущность, или статус 404, если сущность не найдена.
- GET /{Topics|Subtopics|Questions|Answers}/get/{id} получить сущность по её идентификатору. Ожидает целочисленный параметр id. Возвращает статус 200 и сущность, или статус 404, если сущность не найдена.
- GET /{Topics|Subtopics|Questions|Answers}/find поиск сущности по её полю Value. Ожидает строковые параметры в запросе value и match_type и логическим параметром case_sensitivity. Возвращает статус 200 и сущность, или статус 404, если сущность не найдена.
- PUT /{Topics|Subtopics|Questions|Answers}/update/{id} обновить поле Value сущности с указанным идентификатором. Ожидает целочисленный пара-

метр id и строковое тело запроса. Возвращает статус 200 и изменённую сущность, или статус 404, если сущность не найдена.

•DELETE /{Topics|Subtopics|Questions|Answers}/delete/{id} удалить сущность с указанным идентификатором. Ожидает целочисленный параметр id. Возвращает статус 200 и удалённую сущность, или статус 404, если сущность не найдена.

•POST /Answers/add добавить сущность Answer. Ожидает в теле запроса JSON-объект со строковыми полями text и url. Возвращает статус 200 и созданную сущность Answer, или статус 400 с текстом ошибки.

•GET /Answers/get/{id}/questions получить список сущностей Question, относящихся к сущности Answer с указанным идентификатором. Ожидает целочисленный параметр id. Возвращает статус 200 и список сущностей Question, или статус 404, если сущность Answer не найдена.

•POST /Topics/add добавить сущность Topic. Ожидает строковое тело запроса. Возвращает статус 200 и созданную сущность Topic, или статус 400 с текстом ошибки.

•• GET /Topics/get/{id}/subtopics получить список сущностей Subtopic, относящихся к сущности Topic с указанным идентификатором. Ожидает целочисленный параметр id. Возвращает статус 200 и список сущностей Subtopic, или статус 404, если сущность Topic не найдена.

•POST /Subtopics/add добавить сущность Subtopic. Ожидает строковое тело запроса и целочисленный параметр topicId в запросе. Возвращает статус 200 и созданную сущность Subtopic, или статус 400 или 404 с текстом ошибки.

•GET /Subtopics/get/{id}/questions получить список сущностей Question, относящихся к сущности Subtopic с указанным идентификатором. Ожидает целочисленный параметр id. Возвращает статус 200 и список сущностей Question, или статус 404, если сущность Subtopic не найдена.

•GET /Subtopics/get/{id}/topic получить сущность Topic, относящуюся к сущности Subtopic с указанным идентификатором. Ожидает целочисленный

параметр id. Возвращает статус 200 и сущность Topic, или статус 404, если сущность Subtopic не найдена.

- POST /Questions/add добавить сущность Question. Ожидает строковое тело запроса, а также в запросе целочисленные параметры subtopicId и answerId и булев параметр isUiQuestion. Возвращает статус 200 и созданную сущность Question, или статус 400 или 404 с текстом ошибки.
- GET /Questions/get/{id}/subtopic получить сущность Subtopic, относящуюся к сущности Question с указанным идентификатором. Ожидает целочисленный параметр id. Возвращает статус 200 и сущность Topic, или статус 404, если сущность Subtopic не найдена.
- GET /Questions/get/{id}/answer получить сущность Answer, относящуюся к сущности Question с указанным идентификатором. Ожидает целочисленный параметр id. Возвращает статус 200 и сущность Topic, или статус 404, если сущность Subtopic не найдена.
- GET /Questions/get/ui_questions получить сущности Question с полем IsUiQuestion установленном в истинное значение. Не ожидает дополнительных параметров. Возвращает статус 200 и список сущностей Question.
- PUT /Questions/update/{id}/is_ui_question изменить поле IsUiQuestion сущности Question с указанным идентификатором. Ожидаем целочисленный параметр id и булев параметр isUiQuestion в запросе.
- POST /Nlu/Parse выполнить парсинг текста используя текущую загруженные в сервисе NLU модель. Ожидает текстовое тело запроса. Возвращает статус 200 и JSON-объект с результатами парсинга.
- PATCH /Nlu/patch обучить и загрузить модель в сервис NLU. Ожидает параметры в запросе: логические force_training и save_to_default_model_directory, целочисленные augmentation и num_threads, строковый callback_url. Возвращает статус 200 массив байтов, представляющий собой натренированную модель в zip-архиве.

ПРИЛОЖЕНИЕ Б. ПРОГРАММНЫЙ КОД

Весь проект содержит 2133 строк. Сам чат-бот реализован коллегой и составляет 275 строк, здесь он не будет включён. Также не будут включены файлы с объявлениями типов. Остаются файлы реализации контроллеров и веб-сервера, а также дополнительные компоненты для модуля распознавания. Итого получится 1411 строк.

Database/Program.cs

```
using System;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

using System.IO;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using Database.Data;

namespace Database
{
   public class Program
   {
      public static void Main(string[] args)
      {
         var host = CreateHostBuilder(args).Build();

         bool migrateOnStartup = Boolean.Parse(Environment.GetEnvironmentVariable("migrateOnStartup") ?? "false");
         if (migrateOnStartup)
         {
            using (var scope = host.Services.CreateScope())
            {
               var services = scope.ServiceProvider;
               try
               {
                  var Database = services.GetRequiredService<QAContext>().Database;
                  while (!Database.CanConnect()) ;
                  Database.Migrate();
               }
               catch (Exception ex)
               {
                  services
                     .GetRequiredService<ILogger<Program>>()
                     .LogError(ex, "An error occurred while migrating the database.");
               }
            }
         }
         host.Run();
      }
```

```
    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseContentRoot(Directory.GetCurrentDirectory());
                webBuilder.UseStartup<Startup>();
            });
  }
}
```

Database/Startup.cs

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.OpenApi.Models;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using Microsoft.AspNetCore.Identity;
using System;
using System.Text;
using System.Collections.Generic;

using Microsoft.EntityFrameworkCore;
using Database.Data;

namespace Database
{
  public class Startup
  {
    public Startup(IConfiguration configuration)
    {
      Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // конфигурируем сервисы во время выполнения
    public void ConfigureServices(IServiceCollection services)
    {
      services.AddDbContext<QAContext>(options =>
        options.UseNpgsql(Configuration.GetConnectionString("QAContext")));
      services.AddControllers();

      services.AddHttpClient();

      // конфигурируем схему аутентификации
      services.AddAuthentication(options =>
      {
        options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
```

```
        })
        .AddJwtBearer(jwt =>
        {
            var key = Encoding.ASCII.GetBytes(Environment.GetEnvironmentVariable("JWT_TOKEN"));

            jwt.SaveToken = true;
            jwt.TokenValidationParameters = new TokenValidationParameters
            {
                ValidateIssuerSigningKey = true,
                IssuerSigningKey = new SymmetricSecurityKey(key), // Add the secret key to our Jwt encryption
                ValidateIssuer = false,
                ValidateAudience = false,
                RequireExpirationTime = false,
                ValidateLifetime = true
            };
        });

        services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)
                .AddEntityFrameworkStores<QAContext>();

        services.AddSwaggerGen(c =>
        {
            c.SwaggerDoc("v1", new OpenApiInfo { Title = "Database", Version = "v1" });
            c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
            {
                Description = "JWT Authorization header using the Bearer scheme. \r\n\r\nEnter 'Bearer' [space] and then your token in the text input below. \r\n\r\nExample: 'Bearer 12345abcdef'",
                Name = "Authorization",
                In = ParameterLocation.Header,
                Type = SecuritySchemeType.ApiKey,
                Scheme = "Bearer"
            });

            c.AddSecurityRequirement(new OpenApiSecurityRequirement()
            {
                {
                new OpenApiSecurityScheme
                {
                    Reference = new OpenApiReference
                    {
                        Type = ReferenceType.SecurityScheme,
                        Id = "Bearer"
                    },
                    Scheme = "oauth2",
                    Name = "Bearer",
                    In = ParameterLocation.Header,

                    },
                    new List<string>()
                }
            });
        });
    }
```

```
        // конфигурируем обработку HTTP-запросов во время выполнения
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseHsts();
            }

            app.UseSwagger();
            app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "Database v1"));

            app.UseHttpsRedirection();

            app.UseRouting();

            app.UseAuthentication();
            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
}
```

Database/Controllers/EntitiesController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using System.ComponentModel.DataAnnotations;
using System.Text.RegularExpressions;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;

using Database.Data;
using Database.Models.Entities;

namespace Database.Controllers
{
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    public class EntitiesController<C, M, R> : ControllerBase
    where M : Entity
    where C : ControllerBase
    {
        protected readonly ILogger<C> _logger;
```

```
protected readonly QAContext _context;
protected readonly Func<M, R> _unlinker;

protected IQueryable<M> Select()
{
    return _context.Set<M>().OrderBy(e => e.Id);
}

public EntitiesController(QAContext context, ILogger<C> logger, Func<M, R> unlinker)
{
    _context = context;
    _logger = logger;
    _unlinker = unlinker;
}

[HttpGet("get/all")]
public ActionResult<IEnumerable<R>> GetAll(
    [Range(0, Int32.MaxValue)] int? offset = 0,
    [Range(0, 1000)] int? size = 1000
)
{
    return Select()
        .Skip(offset ?? 0)
        .Take(size ?? 1000)
        .Select(_unlinker)
        .ToList();
}

[HttpGet("get/{id}")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
public ActionResult<R> GetById([Required] int id)
{
    try
    {
        var e = Select().First(e => e.Id == id);
        return _unlinker(e);
    }
    catch (Exception)
    {
        return NotFound("Not found");
    }
}

[HttpGet("find")]
[ProducesResponseType(200)]
[ProducesResponseType(400)]
[ProducesResponseType(404)]
public ActionResult<R> Find(
    [Required] string value,
    [RegularExpression("part|full|regex")] string match_type = "part",
    bool? case_sensitivity = false
)
```

```
    {
        var cs = case_sensitivity ?? false;
        match_type ??= "part";
        var string_comparsion = cs ? StringComparison.Ordinal : StringComparison.OrdinalIgnoreCase;
        try
        {
            M e;
            switch (match_type.ToLower())
            {
                case "part":
                    e = Select().First(e => e.Value.Contains(value, string_comparsion));
                    break;
                case "full":
                    e = Select().First(e => e.Value.Equals(value, string_comparsion));
                    break;
                case "regex":
                    e = Select().First(e => Regex.IsMatch(e.Value, value, cs ? RegexOptions.IgnoreCase : RegexOptions.None));
                    break;
                default:
                    return BadRequest("match_type must be 'part' | 'full' | 'regex'");
            }
            return _unlinker(e);
        }
        catch (Exception)
        {
            return NotFound("Not found");
        }
    }

    [HttpPut("update/{id}")]
    [ProducesResponseType(200)]
    [ProducesResponseType(400)]
    [ProducesResponseType(404)]
    public ActionResult<R> Update([FromBody, Required] string new_value, [Required] int id)
    {
        if (Select().Any(e => e.Value == new_value))
        {
            return BadRequest("Already exists");
        }
        try
        {
            var e = Select().First(e => e.Id == id);
            e.Value = new_value;
            _context.SaveChanges();
            return _unlinker(e);
        }
        catch (Exception)
        {
            return NotFound("Not found");
        }
    }
  }
}
```

Database/Controllers/AnswersController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using System.ComponentModel.DataAnnotations;
using Microsoft.EntityFrameworkCore;

using Database.Data;
using Database.Models;
using Database.Models.Entities;
using Database.Domain;

namespace Database.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class AnswersController : EntitiesController<AnswersController, Answer, EntityAnswer>
    {

        public AnswersController(QAContext context, ILogger<AnswersController> logger)
            : base(context, logger, Answer.WithoutReferences)
        {
        }

        [HttpGet("get/{id}/questions")]
        [ProducesResponseType(404)]
        public ActionResult<IEnumerable<EntityQuestion>> GetQuestion(int id)
        {
            try
            {
                return Select()
                    .Include(a => a.Question)
                    .First(a => a.Id == id)
                    .Question
                    .Select(Question.WithoutReferences)
                    .ToList();
            }
            catch (Exception)
            {
                return NotFound("Not found");
            }
        }

        [HttpPost("add")]
        [ProducesResponseType(200)]
        [ProducesResponseType(400)]
        [ProducesResponseType(404)]
        public ActionResult<EntityAnswer> Post([FromBody, Required] AddAnswerRequest addAnswer)
        {
            bool alreadyExists = Select().Any(a => addAnswer.Text.ToLower() == a.Value.ToLower());
```

```
    if (alreadyExists)
    {
        return BadRequest("Already exists");
    }
    var answer = new Answer
    {
        Value = addAnswer.Text,
        FullAnswerUrl = addAnswer.Url,
        Question = new List<Question>()
    };
    _context.Answers.Add(answer);
    _context.SaveChanges();
    return answer.WithoutReferences();
}

[HttpPut("update/{id}/full_answer_url")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
public ActionResult<EntityAnswer> UpdateFullAnswerUrl([Required] int id, [FromBody, Required] string fullAnswerUrl)
{
    try
    {
        var a = Select().First(a => a.Id == id);
        a.FullAnswerUrl = fullAnswerUrl;
        _context.SaveChanges();
        return a.WithoutReferences();
    }
    catch
    {
        return NotFound("Not found");
    }
}

[HttpDelete("delete/{id}")]
[ProducesResponseType(200)]
[ProducesResponseType(400)]
[ProducesResponseType(404)]
public ActionResult<EntityAnswer> Delete(int id)
{
    Answer answer = null;
    try
    {
        answer = Select()
            .Include(a => a.Question)
            .First(a => id == a.Id);
    }
    catch (Exception)
    {
        return NotFound("Not found");
    }
    if (answer.Question.Take(1).Count() > 0)
    {
        return BadRequest("Some questions depends on this answer");
```

```
            }
            _context.Answers.Remove(answer);
            _context.SaveChanges();
            return answer.WithoutReferences();
        }
    }
}
```

Database/Controllers/TopicsController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using System.ComponentModel.DataAnnotations;
using Microsoft.EntityFrameworkCore;

using Database.Data;
using Database.Models;
using Database.Models.Entities;

namespace Database.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class TopicsController : EntitiesController<TopicsController, Topic, EntityTopic>
    {

        public TopicsController(QAContext context, ILogger<TopicsController> logger)
            : base(context, logger, Topic.WithoutReferences)
        {
        }

        [HttpGet("get/{id}/subtopics")]
        [ProducesResponseType(200)]
        [ProducesResponseType(404)]
        public ActionResult<IEnumerable<EntitySubtopic>> GetSubtopics(int id)
        {
            try
            {
                return Select()
                    .Include(t => t.Subtopic)
                    .First(t => t.Id == id)
                    .Subtopic
                    .Select(Subtopic.WithoutReferences)
                    .ToList();
            }
            catch (Exception)
            {
                return NotFound("Not found");
            }
        }
```

```
        [HttpPost("add")]
        [ProducesResponseType(200)]
        [ProducesResponseType(400)]
        public ActionResult<EntityTopic> Post([FromBody, Required] string topicText)
        {
            bool alreadyExists = Select().Any(t => topicText == t.Value);
            if (alreadyExists)
            {
                return BadRequest("Already exists");
            }
            var topic = new Topic
            {
                Value = topicText,
                Subtopic = new List<Subtopic>()
            };
            _context.Topics.Add(topic);
            _context.SaveChanges();
            return topic.WithoutReferences();
        }

        [HttpDelete("delete/{id}")]
        [ProducesResponseType(200)]
        [ProducesResponseType(400)]
        [ProducesResponseType(404)]
        public ActionResult<EntityTopic> Delete(int id)
        {
            Topic topic = null;
            try
            {
                topic = Select()
                    .Include(t => t.Subtopic.Take(1))
                    .First(t => t.Id == id);
            }
            catch (Exception)
            {
                return NotFound("Not found");
            }
            if (topic.Subtopic.Count() > 0)
            {
                return BadRequest("Topic depends on some subtopics. Delete subtopics first");
            }
            _context.Topics.Remove(topic);
            _context.SaveChanges();
            return topic.WithoutReferences();
        }
    }
}
```

Database/Controllers/SubtopicsController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
```

```
using Microsoft.Extensions.Logging;
using System.ComponentModel.DataAnnotations;
using Microsoft.EntityFrameworkCore;

using Database.Data;
using Database.Models;
using Database.Models.Entities;

namespace Database.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class SubtopicsController : EntitiesController<SubtopicsController, Subtopic, EntitySubtopic>
    {

        public SubtopicsController(QAContext context, ILogger<SubtopicsController> logger)
            : base(context, logger, Subtopic.WithoutReferences)
        {
        }

        [HttpGet("get/{id}/topic")]
        [ProducesResponseType(200)]
        [ProducesResponseType(404)]
        public ActionResult<EntityTopic> GetSubtopicTopic(int id)
        {
            try
            {
                return Select()
                    .Include(st => st.Topic)
                    .First(st => st.Id == id)
                    .Topic
                    .WithoutReferences();
            }
            catch (Exception)
            {
                return NotFound("Not found");
            }
        }

        [HttpGet("get/{id}/questions")]
        [ProducesResponseType(200)]
        [ProducesResponseType(404)]
        public ActionResult<IEnumerable<EntityQuestion>> GetQuestions(int id)
        {
            try
            {
                return Select()
                    .Include(st => st.Question)
                    .First(st => st.Id == id)
                    .Question
                    .Select(Question.WithoutReferences)
                    .ToList();
            }
```

```
    catch (Exception)
    {
        return NotFound("Not found");
    }
}

[HttpGet("get/{id}/ui_questions")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
public ActionResult<IEnumerable<EntityQuestion>> GetUiQuestions(int id)
{
    try
    {
        return Select()
            .Include(st => st.Question)
            .First(st => st.Id == id)
            .Question
            .Where(q => q.IsUiQuestion)
            .Select(Question.WithoutReferences)
            .ToList();
    }
    catch (Exception)
    {
        return NotFound("Not found");
    }
}

[HttpPost("add")]
[ProducesResponseType(200)]
[ProducesResponseType(400)]
[ProducesResponseType(404)]
public ActionResult<EntitySubtopic> Post([FromBody, Required] string subtopicText, [Required] int topicId)
{
    bool alreadyExists = Select().Any(st => subtopicText == st.Value);
    if (alreadyExists)
    {
        return BadRequest("Already exists");
    }
    Topic topic = null;
    try
    {
        topic = _context
            .Topics
            .Include(t => t.Subtopic)
            .First(t => topicId == t.Id);
    }
    catch (Exception)
    {
        return NotFound("Topic not found");
    }
    var subtopic = new Subtopic
    {
        Value = subtopicText,
```

```
            TopicId = topicId,
            Topic = topic,
            Question = new List<Question>()
        };
        _context.Subtopics.Add(subtopic);
        _context.SaveChanges();
        topic.Subtopic.Add(subtopic);
        _context.SaveChanges();
        return subtopic.WithoutReferences();
    }

    [HttpDelete("delete/{id}")]
    [ProducesResponseType(200)]
    [ProducesResponseType(400)]
    [ProducesResponseType(404)]
    public ActionResult<EntitySubtopic> Delete(int id)
    {
        Subtopic subtopic = null;
        try
        {
            subtopic = Select()
                .Include(st => st.Question.Take(1))
                .First(st => st.Id == id);
        }
        catch (Exception)
        {
            return NotFound("Subtopic not found");
        }
        if (subtopic.Question.Count() > 0)
        {
            return BadRequest("Subtopic depends on some questions. Delete questions first");
        }
        _context.Subtopics.Remove(subtopic);
        _context.SaveChanges();
        return subtopic.WithoutReferences();
    }
  }
}
```

Database/Controllers/QuestionsController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using System.ComponentModel.DataAnnotations;
using Microsoft.EntityFrameworkCore;

using Database.Data;
using Database.Models;
using Database.Models.Entities;

namespace Database.Controllers
```

```
{
    [ApiController]
    [Route("[controller]")]
    public class QuestionsController : EntitiesController<QuestionsController, Question, EntityQuestion>
    {

        public QuestionsController(QAContext context, ILogger<QuestionsController> logger)
            : base(context, logger, Question.WithoutReferences)
        {
        }

        [HttpGet("get/{id}/subtopic")]
        [ProducesResponseType(200)]
        [ProducesResponseType(404)]
        public ActionResult<EntitySubtopic> GetQuestionSubtopic(int id)
        {
            try
            {
                return Select()
                    .Include(q => q.Subtopic)
                    .First(q => q.Id == id)
                    .Subtopic
                    .WithoutReferences();

            }
            catch (Exception)
            {
                return NotFound("Not found");
            }
        }

        [HttpGet("get/{id}/answer")]
        [ProducesResponseType(200)]
        [ProducesResponseType(404)]
        public ActionResult<EntityAnswer> GetQuestionAnswer(int id)
        {
            try
            {
                return Select()
                    .Include(q => q.Answer)
                    .First(q => q.Id == id)
                    .Answer
                    .WithoutReferences();
            }
            catch (Exception)
            {
                return NotFound("Not found");
            }
        }

        [HttpGet("get/ui_questions")]
        [ProducesResponseType(200)]
        [ProducesResponseType(404)]
```

```
public ActionResult<ICollection<EntityQuestion>> GetUiQuestions()
{
    try
    {
        return Select()
            .Where(q => q.IsUiQuestion)
            .Select(Question.WithoutReferences)
            .ToList();
    }
    catch (Exception)
    {
        return NotFound("Not found");
    }
}

[HttpPost("add")]
[ProducesResponseType(200)]
[ProducesResponseType(400)]
[ProducesResponseType(404)]
public ActionResult<EntityQuestion> Post(
    [FromBody, Required] string questionText,
    [Required] int answerId,
    [Required] int subtopicId,
    bool? isUiQuestion = false
)
{
    bool isUi = isUiQuestion ?? false;
    bool alreadyExists = Select().Any(q => questionText == q.Value);
    if (alreadyExists)
    {
        return BadRequest("Already exists");
    }
    Subtopic subtopic = null;
    try
    {
        subtopic = _context
            .Subtopics
            .Include(st => st.Question)
            .First(subtopic => subtopicId == subtopic.Id);
    }
    catch (Exception)
    {
        return NotFound("Subtopic not found");
    }
    Answer answer = null;
    try
    {
        answer = _context
            .Answers
            .Include(a => a.Question)
            .First(a => a.Id == answerId);
    }
    catch (Exception)
```

```
    {
        return NotFound("Answer not found");
    }
    var question = new Question
    {
        IsUiQuestion = isUi,
        Value = questionText,
        SubtopicId = subtopicId,
        Subtopic = subtopic,
        AnswerId = answerId,
        Answer = answer
    };
    _context.Questions.Add(question);
    _context.SaveChanges();
    answer.Question.Add(question);
    subtopic.Question.Add(question);
    _context.SaveChanges();
    return question.WithoutReferences();
}

[HttpPut("update/{id}/is_ui_question")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
public ActionResult<EntityQuestion> UpdateIsUiQuestion([Required] int id, [FromQuery, Required] bool isUiQuestion)
{
    try
    {
        var q = Select().First(q => q.Id == id);
        q.IsUiQuestion = isUiQuestion;
        _context.SaveChanges();
        return q.WithoutReferences();
    }
    catch
    {
        return NotFound("Not found");
    }
}

[HttpDelete("delete/{id}")]
[ProducesResponseType(200)]
[ProducesResponseType(400)]
[ProducesResponseType(404)]
public ActionResult<EntityQuestion> Delete(int id)
{
    Question question = null;
    try
    {
        question = Select()
            .Include(q => q.Answer)
            .First(q => id == q.Id);
    }
    catch (Exception)
    {
```

```
            return NotFound("Not found");
        }
        _context.Questions.Remove(question);
        _context.SaveChanges();
        question.Answer.Question.Remove(question);
        _context.SaveChanges();
        return question.WithoutReferences();
    }
  }
}
```

Database/Controllers/ NluController.cs

```
using System;
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.WebUtilities;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using System.IO;
using System.Net.Http;
using System.Net.Http.Json;
using System.Linq;
using System.Threading.Tasks;
using YamlDotNet.Serialization;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;

using Database.Data;
using Database.Domain;

namespace Database.Controllers
{
  [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
  [ApiController]
  [Route("[controller]")]
  public class NluController : ControllerBase
  {
    private readonly ILogger<NluController> _logger;
    private readonly QAContext _context;
    private readonly HttpClient client;

    public NluController(QAContext context, ILogger<NluController> logger)
    {
      _context = context;
      _logger = logger;
      client = new HttpClient();
      var nlu_host = Environment.GetEnvironmentVariable("NLU_HOST");
      var nlu_port = Int16.Parse(Environment.GetEnvironmentVariable("NLU_PORT"));
      client.BaseAddress = new UriBuilder("http", nlu_host, nlu_port).Uri;
      client.Timeout = System.Threading.Timeout.InfiniteTimeSpan;
    }
```

```
[HttpGet]
public async Task<string> Get()
{
   var response = await client.GetAsync("/");
   return await response.Content.ReadAsStringAsync();
}

[HttpGet("version")]
public async Task<Nlu.Api.Information.Version> Version()
{
   var response = await client.GetAsync("/version");
   return await response.Content.ReadFromJsonAsync<Nlu.Api.Information.Version>();
}

[AllowAnonymous]
[HttpGet("status")]
public async Task<Nlu.Api.Information.Status> Status()
{
   var response = await client.GetAsync("/status");
   return await response.Content.ReadFromJsonAsync<Nlu.Api.Information.Status>();
}

[HttpGet("domain")]
public async Task<Nlu.Api.Domain.Domain> Domain()
{
   var response = await client.GetAsync("/domain");
   return await response.Content.ReadFromJsonAsync<Nlu.Api.Domain.Domain>();
}

[HttpPost("parse")]
public async Task<ICollection<Nlu.Api.Parse.RestResponse>> Parse([FromBody] ParseRequest message)
{
   var content = JsonContent.Create(message);
   var response = await client.PostAsync("/webhooks/rest/webhook", content);
   return await response.Content.ReadFromJsonAsync<ICollection<Nlu.Api.Parse.RestResponse>>();
}

[HttpPost("test")]
public async Task<Nlu.Api.Test.TestResult> Test(
   [FromQuery] string model = null,
   [FromQuery] string callback_url = null,
   [FromQuery] int? cross_validation_folds = null,
   [FromBody] object common_examples = null
)
{
   string uri = new Uri(client.BaseAddress, "/model/test/intents").ToString();
   if (model != null)
   {
      uri = QueryHelpers.AddQueryString(uri, "model", model);
   }
   if (callback_url != null)
   {
      uri = QueryHelpers.AddQueryString(uri, "callback_url", callback_url);
```

```
    }
    if (cross_validation_folds != null)
    {
        uri = QueryHelpers.AddQueryString(uri, "cross_validation_folds", cross_validation_folds.ToString());
    }

    var content = new Dictionary<string, Dictionary<string, object>>();
    content.Add("rasa_nlu_data", new Dictionary<string, object>());
    content["rasa_nlu_data"].Add("common_examples", common_examples);
    var body = JsonContent.Create(content);

    var response = await client.PostAsync(uri, body);
    return await response.Content.ReadFromJsonAsync<Nlu.Api.Test.TestResult>();
}

[HttpPatch("patch")]
public async Task<byte[]> Patch(
    [FromQuery] bool? save_to_default_model_directory = null,
    [FromQuery] bool? force_training = null,
    [FromQuery] int? augmentation = null,
    [FromQuery] int? num_threads = null,
    [FromQuery] string callback_url = null
)
{
    string uri = new Uri(client.BaseAddress, "/model/train").ToString();
    if (save_to_default_model_directory != null)
    {
        uri = QueryHelpers.AddQueryString(uri, "save_to_default_model_directory", save_to_default_model_directory.ToString());
    }
    if (force_training != null)
    {
        uri = QueryHelpers.AddQueryString(uri, "force_training", force_training.ToString());
    }
    if (augmentation != null)
    {
        uri = QueryHelpers.AddQueryString(uri, "augmentation", augmentation.ToString());
    }
    if (num_threads != null)
    {
        uri = QueryHelpers.AddQueryString(uri, "num_threads", num_threads.ToString());
    }
    if (callback_url != null)
    {
        uri = QueryHelpers.AddQueryString(uri, "callback_url", callback_url);
    }

    Nlu.Config Config;
    using (var reader = new StreamReader(Environment.GetEnvironmentVariable("NLU_CONFIG_FILE")))
    {
        Config = new DeserializerBuilder().Build().Deserialize<Nlu.Config>(reader.ReadToEnd());
    }
    Config.intents = _context.Answers.Select(a => $"a{a.Id}").ToList();
    Config.nlu = _context
```

```
    .Questions
    .ToList()
    .Select(q => new
    {
        Intent = $"a{q.AnswerId}",
        Example = q.Value
    })
    .GroupBy(i => i.Intent)
    .Select(i => new Nlu.Config.Intent
    {
        intent = i.Key,
        examples = String.Join("\n", i.Select(q => $"- {q.Example}"))
    });

var responses = _context
    .Answers
    .ToList()
    .Select(a => new
    {
        Action = $"utter_a{a.Id}",
        Answer = a.Value
    })
    .GroupBy(r => r.Action)
    .ToDictionary(
        r => r.Key,
        r => r.Select(i => new Nlu.Config.Response { text = i.Answer })
    );
Config.responses = Config
    .responses
    .Concat(responses)
    .GroupBy(r => r.Key, r => r.Value)
    .ToDictionary(r => r.Key, r => r.First());

var rules = _context
    .Answers
    .Select(a => new Nlu.Config.Rule
    {
        rule = $"a{a.Id}",
        steps = new List<object> {
                new { intent =  $"a{a.Id}" },
                new { action =  $"utter_a{a.Id}" }
        }
    });
Config.rules = Config.rules.Concat(rules);

var Serializer = new SerializerBuilder().Build();
var yaml = Serializer.Serialize(Config);
var body = new StringContent(yaml, System.Text.Encoding.UTF8, "application/x-yaml");
body.Headers.ContentType = new System.Net.Http.Headers.MediaTypeHeaderValue("application/x-yaml");
var response = await client.PostAsync(uri, body);
var model = response.Content.ReadAsByteArrayAsync();
var model_file = response.Headers.GetValues("filename").First();
response = await client.PutAsync("/model", JsonContent.Create(new Dictionary<string, string> {
```

```
            { "model_file", "models/" + model_file }
        }));
        var m = await model;
        return m;
      }
    }
}
```

Database/Controllers/AuthManagementController.cs

```
using System;
using System.IO;
using System.Text;
using System.Collections.Generic;
using System.Linq;
using System.ComponentModel.DataAnnotations;
using System.Threading.Tasks;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Options;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;

using Database.Domain;

namespace Database.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class AuthManagementController : ControllerBase
    {
        private readonly UserManager<IdentityUser> _userManager;
        private readonly string _jwt_token;

        public AuthManagementController(UserManager<IdentityUser> userManager)
        {
            _userManager = userManager;
            _jwt_token = Environment.GetEnvironmentVariable("JWT_TOKEN");
        }

        [HttpGet("RegisterFirst")]
        [AllowAnonymous]
        [ProducesResponseType(200)]
        [ProducesResponseType(400)]
        [ProducesResponseType(500)]
        public async Task<ActionResult> RegisterFirst()
        {
            var email = Environment.GetEnvironmentVariable("FIRST_EMAIL");
            // check i the user with the same email exist
            var existingUser = await _userManager.FindByEmailAsync(email);
```

```
    if (existingUser != null)
    {
        return BadRequest("Email already exist");
    }

    var password = Environment.GetEnvironmentVariable("FIRST_PASSWORD");
    var newUser = new IdentityUser() { Email = email, UserName = email };
    var isCreated = await _userManager.CreateAsync(newUser, password);
    if (isCreated.Succeeded)
    {
        return Ok();
    }
    return StatusCode(500, isCreated.Errors.ToList());
}

[HttpPost("Register")]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
[ProducesResponseType(200)]
[ProducesResponseType(400)]
[ProducesResponseType(500)]
public async Task<ActionResult<AuthResult>> Register([FromBody, Required]AuthRequest user)
{
    // check i the user with the same email exist
    var existingUser = await _userManager.FindByEmailAsync(user.Email);

    if (existingUser != null)
    {
        return BadRequest("Email already exist");
    }

    var newUser = new IdentityUser() { Email = user.Email, UserName = user.Email };
    var isCreated = await _userManager.CreateAsync(newUser, user.Password);
    if (isCreated.Succeeded)
    {
        var jwtToken = GenerateJwtToken(newUser, out int lifetime);

        return Ok(new AuthResult()
        {
            Token = jwtToken,
            Expires = lifetime,
        });
    }
    return StatusCode(500, isCreated.Errors.ToList());
}

private string GenerateJwtToken(IdentityUser user, out int lifetime)
{
    // Now its ime to define the jwt token which will be responsible of creating our tokens
    var jwtTokenHandler = new JwtSecurityTokenHandler();

    // We get our secret from the appsettings
    var key = Encoding.ASCII.GetBytes(_jwt_token);
```

```
    // we define our token descriptor
    // We need to utilise claims which are properties in our token which gives information about the token
    // which belong to the specific user who it belongs to
    // so it could contain their id, name, email the good part is that these information
    // are generated by our server and identity framework which is valid and trusted
    lifetime = Int32.Parse(Environment.GetEnvironmentVariable("TOKEN_LIFETIME"));
    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity(new[]
        {
            new Claim("Id", user.Id),
            new Claim(JwtRegisteredClaimNames.Sub, user.Email),
            new Claim(JwtRegisteredClaimNames.Email, user.Email),
            new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
        }),
        Expires = DateTime.UtcNow.AddSeconds(lifetime),
        // here we are adding the encryption alogorithim information which will be used to decrypt our token
        SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha512Signature)
    };

    var token = jwtTokenHandler.CreateToken(tokenDescriptor);

    return jwtTokenHandler.WriteToken(token);
}

[AllowAnonymous]
[HttpPost("Login")]
[ProducesResponseType(200)]
[ProducesResponseType(400)]
[ProducesResponseType(404)]
public async Task<ActionResult<AuthResult>> Login([FromBody, Required]AuthRequest user)
{
    IdentityUser existingUser;
    // check if the user with the same email exist
    try
    {
        existingUser = await _userManager.FindByEmailAsync(user.Email);
    }
    catch (Exception)
    {
        return NotFound();
    }

    if (existingUser == null)
    {
        // We dont want to give to much information on why the request has failed for security reasons
        return BadRequest("Invalid authentication request");
    }

    // Now we need to check if the user has inputed the right password
    var isCorrect = await _userManager.CheckPasswordAsync(existingUser, user.Password);

    if (isCorrect)
```

```
            {
                var jwtToken = GenerateJwtToken(existingUser, out int lifetime);

                return Ok(new AuthResult()
                {
                    Token = jwtToken,
                    Expires = lifetime
                });
            }
            else
            {
                // We dont want to give to much information on why the request has failed for security reasons
                return BadRequest("Invalid authentication request");
            }
        }
    }
}
```

NLU/spelling_correction.py

```
import typing
from typing import Any, Optional, Text, Dict, List, Type

import os

from rasa.nlu.components import Component
from rasa.nlu.config import RasaNLUModelConfig
from rasa.shared.nlu.training_data.training_data import TrainingData
from rasa.shared.nlu.training_data.message import Message

if typing.TYPE_CHECKING:
    from rasa.nlu.model import Metadata

import jamspell

class SpellingCorrector(Component):
    """Spelling corrector"""

    # Which components are required by this component.
    # Listed components should appear before the component itself in the pipeline.
    @classmethod
    def required_components(cls) -> List[Type[Component]]:
        """Specify which components need to be present in the pipeline."""
        return []

    corrector = None

    # Defines the default configuration parameters of a component
    # these values can be overwritten in the pipeline configuration
    # of the model. The component should choose sensible defaults
    # and should be able to create reasonable results with the defaults.
    defaults = {}

    # Defines what language(s) this component can handle.
```

```
# This attribute is designed for instance method: `can_handle_language`.
# Default value is None which means it can handle all languages.
# This is an important feature for backwards compatibility of components.
supported_language_list = None

# Defines what language(s) this component can NOT handle.
# This attribute is designed for instance method: `can_handle_language`.
# Default value is None which means it can handle all languages.
# This is an important feature for backwards compatibility of components.
not_supported_language_list = None

def __init__(self, component_config: Optional[Dict[Text, Any]] = None) -> None:
    super().__init__(component_config)
    self.corrector = jamspell.TSpellCorrector()
    JAMSPELL_FOLDER = os.environ["JAMSPELL_FOLDER"]
    SPELLING_MODEL_FILE = os.environ["SPELLING_MODEL_FILE"]
    model_file_path = os.path.join(JAMSPELL_FOLDER, SPELLING_MODEL_FILE)
    self.corrector.LoadLangModel(model_file_path)

def train(
    self,
    training_data: TrainingData,
    config: Optional[RasaNLUModelConfig] = None,
    **kwargs: Any,
) -> None:
    """Train this component.

    This is the components chance to train itself provided
    with the training data. The component can rely on
    any context attribute to be present, that gets created
    by a call to :meth:`components.Component.pipeline_init`
    of ANY component and
    on any context attributes created by a call to
    :meth:`components.Component.train`
    of components previous to this one."""
    pass

def process(self, message: Message, **kwargs: Any) -> None:
    """Process an incoming message.

    This is the components chance to process an incoming
    message. The component can rely on
    any context attribute to be present, that gets created
    by a call to :meth:`components.Component.pipeline_init`
    of ANY component and
    on any context attributes created by a call to
    :meth:`components.Component.process`
    of components previous to this one."""

    message.data["text"] = self.corrector.FixFragment(message.data["text"])

    pass
```

```
    def persist(self, file_name: Text, model_dir: Text) -> Optional[Dict[Text, Any]]:
        """Persist this component to disk for future loading."""
        return { "spelling_model_file": os.path.join(os.environ["JAMSPELL_FOLDER"], os.environ["SPELLING_MODEL_FILE"]) }

    @classmethod
    def load(
        cls,
        meta: Dict[Text, Any],
        model_dir: Text,
        model_metadata: Optional["Metadata"] = None,
        cached_component: Optional["Component"] = None,
        **kwargs: Any,
    ) -> "Component":
        """Load this component from file."""

        if cached_component:
            return cached_component
        else:
            return cls(meta)
```

NLU/text_preprocessing.py

```
import typing
from typing import Any, Optional, Text, Dict, List, Type

import re

from rasa.nlu.components import Component
from rasa.nlu.config import RasaNLUModelConfig
from rasa.shared.nlu.training_data.training_data import TrainingData
from rasa.shared.nlu.training_data.message import Message

if typing.TYPE_CHECKING:
    from rasa.nlu.model import Metadata

class TextPreprocessor(Component):
    """A new component"""

    # Which components are required by this component.
    # Listed components should appear before the component itself in the pipeline.
    @classmethod
    def required_components(cls) -> List[Type[Component]]:
        """Specify which components need to be present in the pipeline."""
        return []

    # Defines the default configuration parameters of a component
    # these values can be overwritten in the pipeline configuration
    # of the model. The component should choose sensible defaults
    # and should be able to create reasonable results with the defaults.
    defaults = {}

    # Defines what language(s) this component can handle.
    # This attribute is designed for instance method: `can_handle_language`.
```

```
# Default value is None which means it can handle all languages.
# This is an important feature for backwards compatibility of components.
supported_language_list = None

# Defines what language(s) this component can NOT handle.
# This attribute is designed for instance method: `can_handle_language`.
# Default value is None which means it can handle all languages.
# This is an important feature for backwards compatibility of components.
not_supported_language_list = None

def __init__(self, component_config: Optional[Dict[Text, Any]] = None) -> None:
    super().__init__(component_config)

def train(
    self,
    training_data: TrainingData,
    config: Optional[RasaNLUModelConfig] = None,
    **kwargs: Any,
) -> None:
    """Train this component.

    This is the components chance to train itself provided
    with the training data. The component can rely on
    any context attribute to be present, that gets created
    by a call to :meth:`components.Component.pipeline_init`
    of ANY component and
    on any context attributes created by a call to
    :meth:`components.Component.train`
    of components previous to this one."""
    pass

def process(self, message: Message, **kwargs: Any) -> None:
    """Process an incoming message.

    This is the components chance to process an incoming
    message. The component can rely on
    any context attribute to be present, that gets created
    by a call to :meth:`components.Component.pipeline_init`
    of ANY component and
    on any context attributes created by a call to
    :meth:`components.Component.process`
    of components previous to this one."""

    text = message.data["text"]
    message.data["originalText"] = text
    text = re.sub(r"[^a-яA-ЯёЁ\d\s]", " ", text.lower().strip())
    message.data["text"] = re.sub(r"\s+", " ", text.strip())

def persist(self, file_name: Text, model_dir: Text) -> Optional[Dict[Text, Any]]:
    """Persist this component to disk for future loading."""
    pass

@classmethod
```

```
def load(
    cls,
    meta: Dict[Text, Any],
    model_dir: Text,
    model_metadata: Optional["Metadata"] = None,
    cached_component: Optional["Component"] = None,
    **kwargs: Any,
) -> "Component":
    """Load this component from file."""

    if cached_component:
        return cached_component
    else:
        return cls(meta)
```