СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc107482452)

[ВВЕДЕНИЕ 3](#_Toc107482453)

[1 Описание предметной области 4](#_Toc107482454)

[1.1 Аналитический обзор предметной области 4](#_Toc107482455)

[1.2 Аналитический обзор существующих аналогов 6](#_Toc107482456)

[2 Постановка задачи 8](#_Toc107482457)

[3 Проектирование базы данных 9](#_Toc107482458)

[3.1 Основные сущности 9](#_Toc107482459)

[3.2 Основные связи 9](#_Toc107482460)

[3.3 Отношения 9](#_Toc107482461)

[3.4 Атрибуты и их домены 9](#_Toc107482462)

[3.5 Правила целостности по сущностям и по ссылкам 10](#_Toc107482463)

[3.6 Схема базы данных 10](#_Toc107482464)

[3.7 Тестовые данные базы данных 11](#_Toc107482465)

[4 Программное обеспечение системы 12](#_Toc107482466)

[4.1 Программный интерфейс 12](#_Toc107482467)

[4.2 Выбор средств для разработки приложения 12](#_Toc107482468)

[4.3 Выбор СУБД 12](#_Toc107482469)

[5 Описание функционирования системы 14](#_Toc107482470)

[5.1 Страница авторизации 14](#_Toc107482471)

[5.2 Страница со списком категорий 15](#_Toc107482472)

[5.3 Страница с детальной информацией о категории 15](#_Toc107482473)

[5.4 Страница со списком алгоритмов 15](#_Toc107482474)

[5.5 Страница с детальной информацией об алгоритме 16](#_Toc107482475)

[5.6 Страница со списком задач 16](#_Toc107482476)

[5.7 Страница с детальной информацией о задаче 17](#_Toc107482477)

[ЗАКЛЮЧЕНИЕ 18](#_Toc107482478)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 19](#_Toc107482479)

[ПРИЛОЖЕНИЕ А 20](#_Toc107482480)

[ПРИЛОЖЕНИЕ Б 30](#_Toc107482481)

ВВЕДЕНИЕ

Основной тип данных, с которым люди постоянно работают является неструктурированный текст. Для людей извлечение данных из таких текстов не является проблемой, поскольку человек может определять смысл по контексту или догадываться о смысле по каким-либо косвенным признаком. Однако для машины такие действия не являются очевидными, и обработка таких текстов с помощью компьютеров вызывает затруднения.

Для решения данной проблемы появилась технология Natural Language Processing (NLP), которая отвечает за обработку, структуризация и извлечению данных из естественного для человека текста. Технологии NLP в данный момент используются во многих сферах: наука, бизнес, обучение. Если раньше данная технология была больше теоретическая и использовалась в узком количестве областей, то сейчас NLP используется для решения большого количества бизнес-задач, а поскольку бизнес заинтересован в данной технологии, то она очень быстро развивается. Создается большое количество различных алгоритмов для решения разнообразных задач в области обработки естественного языка. Из-за очень быстрого развития данного направления разработчикам необходимо постоянно быть в курсе новейших алгоритмах NLP, чтобы использовать наиболее эффективные алгоритмы и приносить бизнесу наибольшую прибыль. Для решения данной проблемы необходимо иметь постоянно обновляющуюся базу знаний актуальных алгоритмов обработки естественного языка.

# 1 Описание предметной области

## 1.1 Аналитический обзор предметной области

Главной особенностью обработки естественного языка является то, что хоть формулировки задач являются достаточно простыми, однако главная проблема заключается в неоднозначности естественного языка: в нем присутствуют омонимы, синонимы, многозначные слова. Основная задача данной области заключается в обучении компьютера решать данные неоднозначности и определять верный смысл.

Основные задачи, которые решает NLP [1]:

1. машинный перевод;
2. классификация текстов;
3. извлечение именованных сущностей;
4. извлечение фактов;
5. поиск обоснования в тексте;
6. диалоговые системы.

Если раньше данные задачи зачастую решались с помощью классических методов машинного обучения, то сейчас все чаще для подобных задач используются нейросети, поскольку они способны дать более точный результат.

Особенности процесса обработки естественного языка:

1. для большинства задач обработки атомарной единицей, по которой вычисляются признаки является слово (токен);
2. для токена определяется контекст, которым чаще всего выступает предложение;
3. большинство задач решается на уровне предложения, однако бывают исключения. Например, для диалоговых систем необходимо учитывать контекст между предложениями;
4. основа любого алгоритма обработки текста на естественном языке – сегментация (разделение текста на отдельные предложения) и токенизация (разделения сегментов на отдельные слова).

Для всех выделенных токенов определяются признаки, которые являются контекстно независимыми. Такими признаками являются:

1. эмбеддинги;
2. символьные признаки;
3. дополнительные признаки.

Эмбеддинг – это сопоставление произвольной сущности (например, узла в графе или кусочка картинки) некоторому вектору [2].

Суть эмбеддинга заключается в сжатом представлению о контексте токена.

Использование эмбеддингов удобно, поскольку размерность таких векторов гораздо меньше, чем размерность дискретного вектора, который может достигать несколько сотен тысяч на больших данных. Помимо этого, при дискретном векторе слова представляются в виде списка, обычно в лексикографическом порядке, а при использовании эмбеддинга слова располагаются в зависимости от контекста и с такими данными удобнее работать при анализе.

На рисунке 1.1 представлен пример обученной модели эмбеддинга.

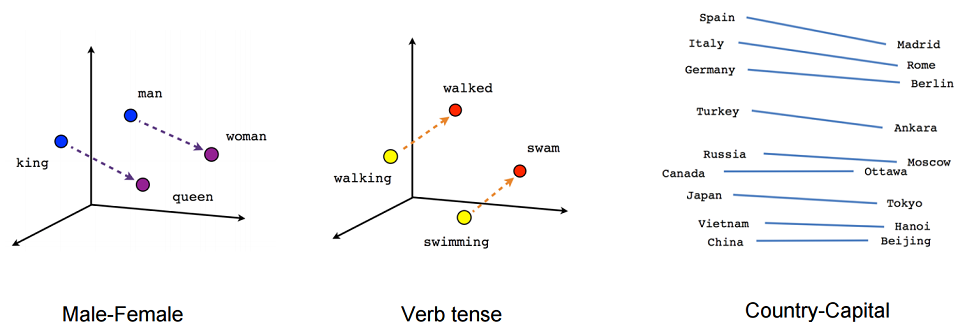


Рисунок 1.1 – Пример обученной модели эмбеддинга

На рисунке видно, что модели эмбеддингов могут улавливать семантические и синтаксические связи разных слов. Например, модель распознает, что король и королева соотносятся также, как мужчина и женщина.

Еще одной моделью векторизации является «сумка слов» (bag of word), которая представляет из себя вектор длиной в словарь и для каждого слова высчитывается количество вхождений данного слова в анализируемый текст. После этого полученное число вставляется в соответствующее место в векторе.

Хоть эти модели являются одними из самых простых моделей, однако они до сих пор используются во многих алгоритмах извлечения данных из текста.

Одной из самых популярных задач NLP является извлечение именованных сущностей (Named Entity Recognition, NER). Задача NER выделить спаны сущностей в тексте (спан – непрерывный фрагмент текста) [3].

Именованными сущностями являются персоны, организации, локации, даты, прочие именованные сущности.

Большинство существующих моделей NER основано на алгоритме CRF (Conditional random field) – разновидность метода Марковский случайных полей [4].

## 1.2 Аналитический обзор существующих аналогов

Аналогом разрабатываемой системы является ресурс paperswithcode.com, цель которого заключается в хранении материалов о машинном обучении. В данном веб-приложении хранятся описания методов, применяемых в машинном обучении, примеры кода, ссылки на реализованные проекты и другая справочная информация.

Пример страницы с информацией об одной из технологии извлечения данных из текста представлен на рисунке 1.2.

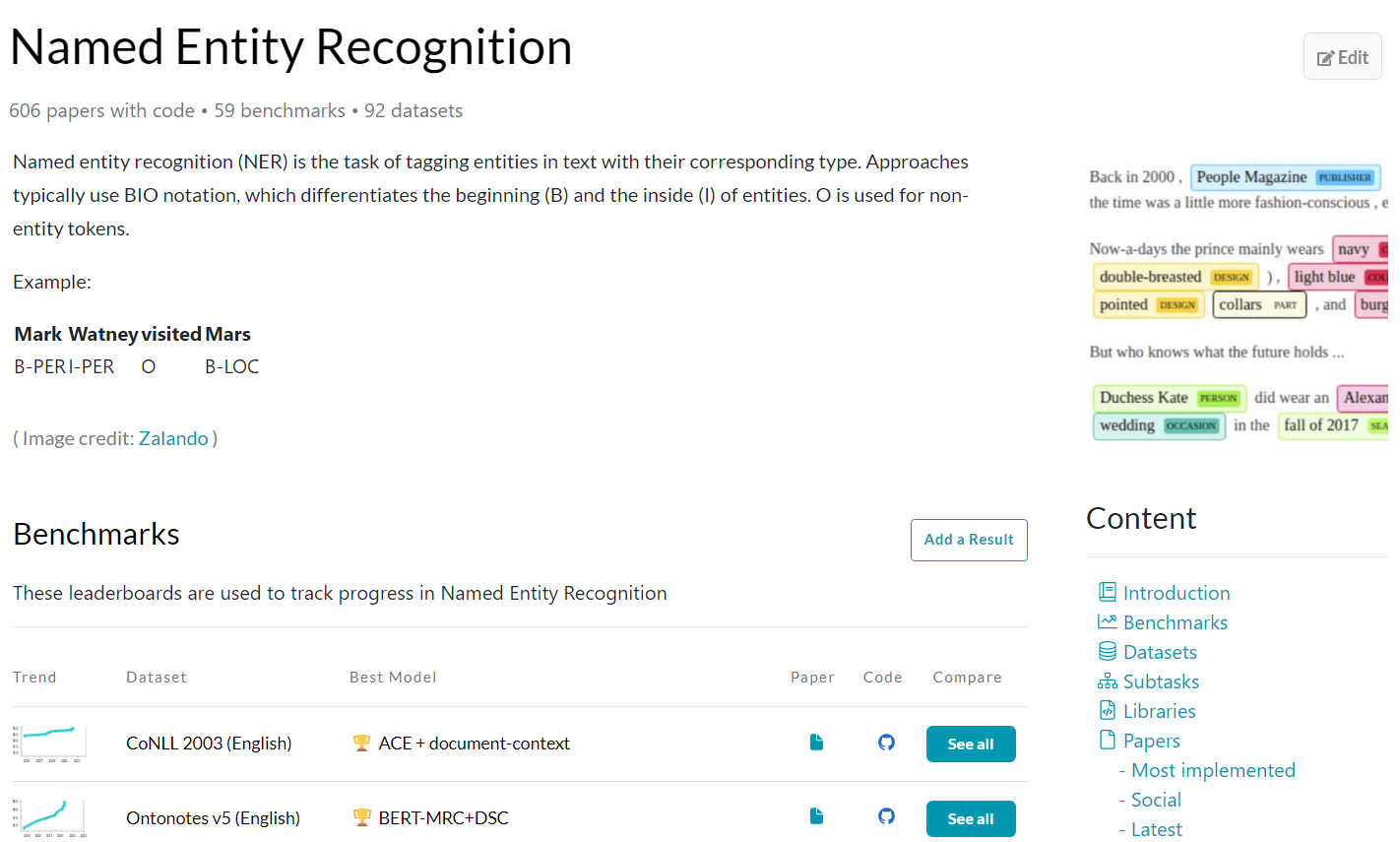


Рисунок 1.2 – Пример страницы paperswithcode.com

У каждой статьи есть метки категорий и задач, при переходе по которым пользователь может найти информацию о технологиях и алгоритмах, которые относятся к той же самой категории или решает схожую задачу.

У каждого алгоритма есть список с примерами реализаций, опубликованными на github.

Помимо этого, на странице есть результаты бэнчмарков, которые отражают эффективность данного алгоритма.

Вся информация, предоставленная на сайте, лицензирована с помощью CC-BY-SA и каждый пользователь может свободно использовать всю опубликованную информацию, а также предлагать изменения в существующих материалах (как в Википедии).

Главным недостатком данного приложения является то, что он является полностью англоязычным и не носителям английского языка может быть сложно изучать информацию, предоставленную на сайте. Если обычный текст можно переводить с помощью машинного переводчика, например Google Translate, то сложный технический текст зачастую переводится не совсем корректно из-за специфических терминов и лексики. Помимо этого, данный сервис охватывает информацию о всех аспектах машинного обучения и обработки данных. На сайте есть информация об обработке изображений, аудиоинформации и прочих видах данных. Поскольку сервис охватывает все аспекты работы с данными, то для пользователя, который хочет ознакомиться исключительно с информацией об извлечении данных из текста, в данном приложении будет много избыточных данных.

# 2 Постановка задачи

Цель данной работы является изучение процесса извлечения данных из текста и разработка системы для хранения информации о технологиях извлечения данных из текста и их реализованных алгоритмах. Система должна предоставлять пользователям справочную информацию о технологии, способах ее применения, а также примеры реализованных инструментов данных технологий. Пользователь должен иметь возможность поиска нужного ему алгоритма по названию или фильтрации списка алгоритмов по решаемым ими задачам. Разрабатываемая система должна быть представлена в виде веб-приложения.

Функциональные требования к разрабатываемой системе.

1. Функция просмотра категорий и алгоритмов, которые относятся к данной категории.
2. Функция просмотра списка задач и алгоритмов, которые решают данную задачу.
3. Функция просмотра детальной информации о каждом алгоритме.
4. Функция поиска алгоритмов по названию.

# 3 Проектирование базы данных

На основании анализа предметной области выделим сущности, определим список атрибутов и опишем связи между сущностями.

## 3.1 Основные сущности

Из анализа предметной области можно выделить сущности и их атрибуты.

1. АЛГОРИТМ (Название, Контент, Ссылка, Дата создания, Дата обновления, Создано Кем, Обновлено Кем).
2. ЗАДАЧА (Название, Описание).
3. КАТЕГОРИЯ (Название, Описание).

## 3.2 Основные связи

В ходе проектирования были выявлены следующие связи:

1. связь «относится к/содержит» между сущностями АЛГОРИТМ и КАТЕГОРИЯ. Связь «один-ко-многим»;
2. связь «решает/решается» между сущностями АЛГОРИТМ и ЗАДАЧА. Связь «многие-ко-многим»;
3. связь «решает/решается» между сущностями ЗАДАЧА и КАТЕГОРИЯ. Связь «многие-ко-многим».

## 3.3 Отношения

На основе выделенных сущностей и связей были определены следующие отношения:

1. *АЛГОРИТМ (Id, Название, Контент, Ссылка, Id категории, Дата создания, Кем создано, Дата обновления, Кем обновлено);*
2. *ЗАДАЧА (Id, название, Описание);*
3. *АЛГОРИТМ ЗАДАЧИ (Id алгоритма, Id задачи);*
4. *КАТЕГОРИЯ (Id категории, Название, Описание);*
5. *КАТЕГОРИЯ ЗАДАЧИ (Id категории, Id задачи);*

## 3.4 Атрибуты и их домены

Для атрибутов необходимо определить домены

Dom (Имя категории) = {x | STRING(x) & Длина(x) <= 100}. Самое длинное название категории не превышает 100 символов. В большинстве случаев название состоит максимум из нескольких слов. Обязательное значение.

Dom (Описание категории) = {x| STRING(x) & Длина(x) <= 3000}. Описания категорий не превышают 3000 символов. Обязательное значение.

Dom (Название задачи) = {x | STRING(x) & Длина(x) <= 100}. Самое длинное название задачи не превышает 100 символов. Обязательное значение.

Dom (Описание задачи) = {x| STRING(x) & Длина(x) <= 3000}. Описания категорий не превышают 2000 символов. Обязательное значение.

Dom (Имя алгоритма) = {x | STRING(x) & Длина(x) <= 150}. Самое длинное название алгоритма не превышает 100 символов. Обязательное значение.

Dom (Контент алгоритма) = {x| STRING(x) & Длина(x) <= 3000}. Длина блока с описанием алгоритма не превышает 3000 символов. Обязательное значение.

Dom (Ссылка на реализацию алгоритма) = {x| STRING(x) & Длина(x) <= 150}. Обязательное значение.

## 3.5 Правила целостности по сущностям и по ссылкам

Целостность по сущностям в данной системе сохраняется, поскольку каждый кортеж отношений отличается от любого другого кортежа этого отношения.

Целостность по ссылкам.

1. Отношение АЛГОРИТМ ЗАДАЧИ в качестве составного ключа имеет два внешних ключа: Id алгоритма, который является ключом отношения АЛГОРИТМ и Id задачи, который является ключом отношения ЗАДАЧА.
2. Отношение КАТЕГОРИЯ ЗАДАЧИ в качестве составного ключа имеет два внешних ключа: Id категории, который является ключом отношения КАТЕГОРИЯ и Id задачи, который является ключом отношения ЗАДАЧА.

## 3.6 Схема базы данных

На рисунке 3.1 представлена схема физической модели базы данных, построенная с помощью инструментов Microsoft SQL Server Management Studio и фреймворка Entity Framework.

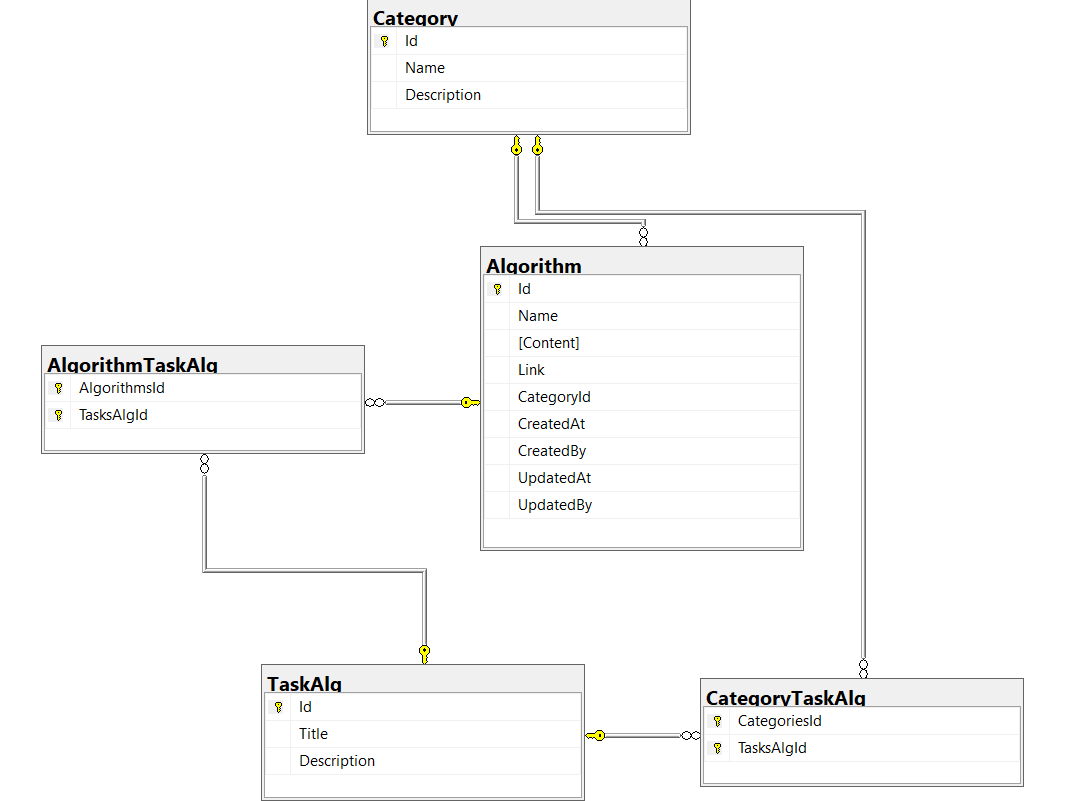


Рисунок 3.1 – Схема физической модели базы данных

Миграции для создания базы данных находятся в приложении Б.

## 3.7 Тестовые данные базы данных

Для тестирования системы необходимо заполнить базу данных тестовыми данными. Код для заполнения базы данных при помощи Entity Framework находится в приложении Б.

# 4 Программное обеспечение системы

## 4.1 Программный интерфейс

В качестве программного интерфейса для работы программы выберем веб-приложение, поскольку для использования веб-приложения пользователю не требуется дополнительная установка приложения на свой компьютер. При разработке реальной системы веб-приложение будет самым оптимальным выбором, поскольку для поиска какого-либо алгоритма пользователю гораздо удобнее просто зайти на сайт, чем скачивать полноценное приложение и устанавливать его.

## 4.2 Выбор средств для разработки приложения

Определившись с типом приложения, необходимо выбрать средства для его разработки. Рассмотрим самые популярные из них:

1. язык программирования JavaScript + HTML+ CSS. Преимущество данного варианта заключается в том, что JavaScript поддерживает большинство браузеров и риск возникновения каких-либо конфликтов минимален;
2. язык программирования C# + ASP.NET. Подходит как для разработки большой группой разработчиков, так и для разработки несколькими разработчиками, поскольку упрощает управление сложностью, разделяя приложение на модель, представление, контроллеры. Помимо этого, .NET Core является кроссплатформенной средой разработки с открытым исходным кодом.
3. язык программирования PHP + HTML. Преимущество PHP состоит в том, что он распространяется со свободной лицензией, а также прост для начального освоения. Помимо этого, он хорошо работает с MySQL.

Для разработки системы была выбрана связка C# + ASP.NET, поскольку данную связку удобно использовать в системах, которые предполагают расширение функционала, а также тот факт, что приложение будет являться кроссплатформенным и его можно будет запустить на сервере с любой операционной системой. В качестве дополнительных средств для разработки был использован фреймворк Bootstrap [5]. Разработка системы будет осуществлять в среде Visual Studio.

## 4.3 Выбор СУБД

Для выбора СУБД рассмотрим самые популярные SQL-базы данных, используемые в настоящее время для разработки:

1. PostgreSQL. Главные особенности заключаются в поддержке табличных пространств, хранимых процедур, объединений, представлений и триггеров;
2. Microsoft SQL Server. Главные особенности заключаются в высокой производительности и возможности установить разные версии на одном компьютере, также преимуществом является то, что C# имеет встроенные инструменты для работы с данной СУБД, что облегчает разработку и интеграцию базы данных в приложение.
3. MySQL. Главные особенности заключаются в бесплатной лицензии, легкости использования и масштабируемостью.

В качестве СУБД была выбрана Microsoft SQL Server, поскольку данная СУБД имеет встроенную поддержку с C# и ASP.NET в частности. Разработка базы данных будет осуществляться в Microsoft SQL Server Management Studio.

# 5 Описание функционирования системы

Для текущего варианта прототипа приложения были выделены следующие основные страницы:

1. страница авторизации;
2. страница со списком категорий;
3. страница с детальной информацией о категории;
4. страница со списком задач;
5. страница с детальной информацией о задаче;
6. страница со списком алгоритмов;
7. страница с детальной информацией об алгоритме.

Переходы для основных страниц находятся в «шапке» сайта. На рисунке 5.1 представлен внешний вид «шапки» веб-приложения.



Рисунок 5.1 – «Шапка» веб-приложения

## 5.1 Страница авторизации

При попытке изменения существующих данных в системе приложение должно проверить, авторизирован ли пользователь. Если нет, тогда произойдет перенаправление на страницу авторизации. Пользователь не сможет получить доступ к изменению данных, пока не авторизуется в приложении. Код страницы находится в приложении А. Пример работы представлен на рисунке 5.2.

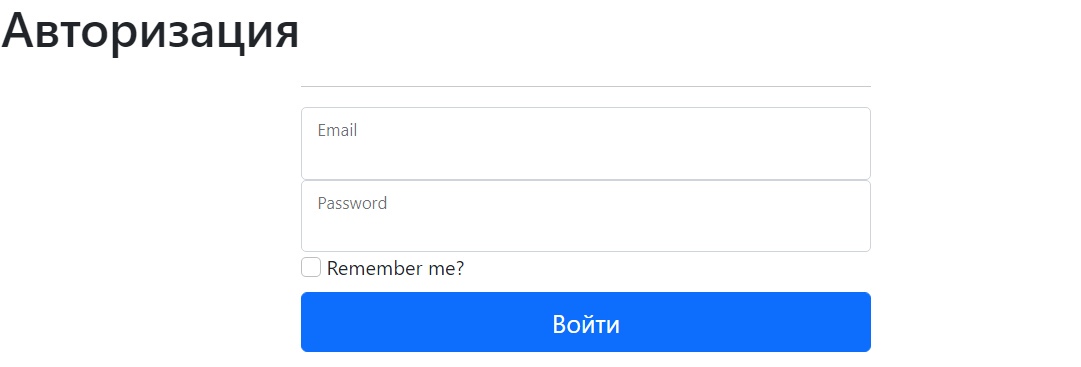


Рисунок 5.2 – Страница авторизации

## 5.2 Страница со списком категорий

На данной странице пользователь может ознакомиться со списком всех категорий и перейти на страницу детальной информации нужной категории. Код страницы находится в приложении А. На рисунке 5.3 представлен пример списка категорий.

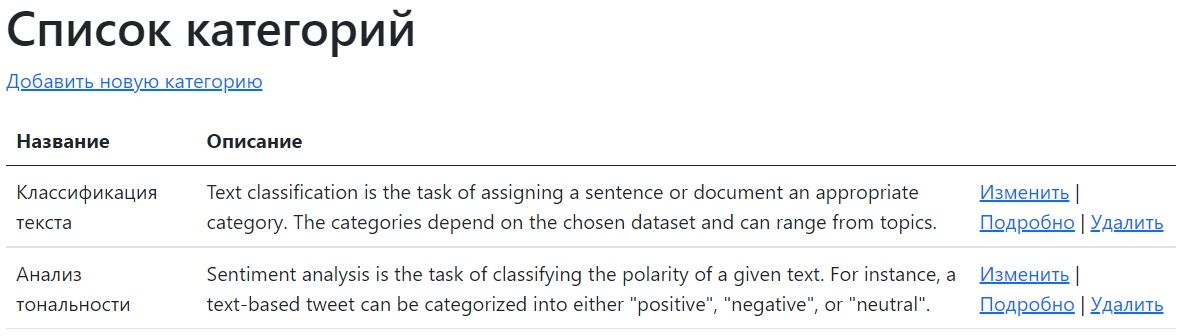


Рисунок 5.3 – Список категорий

## 5.3 Страница с детальной информацией о категории

На данной странице пользователь может ознакомиться с детальной информацией о категории и перейти на страницу списка алгоритмов у данной категории. Код страницы находится в приложении А. На рисунке 5.4 представлен пример страницы с детальной информацией о категории.

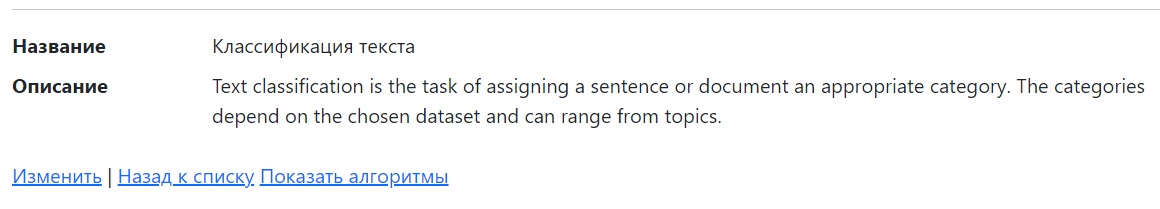


Рисунок 5.4 – Страница детальной информации о категории

## 5.4 Страница со списком алгоритмов

На данной странице пользователь может ознакомиться со списком алгоритмов, которые относятся к выбранной категории. Код страницы находится в приложении А. На рисунке 5.5 представлен пример страницы со списком алгоритмов.



Рисунок 5.5 – Страница со списком алгоритмов

## 5.5 Страница с детальной информацией об алгоритме

На данной странице пользователь может ознакомиться с подробной информацией о выбранном алгоритме. Код страницы находится в приложении А. На рисунке 5.6 представлен пример страницы с детальной информацией об алгоритме.

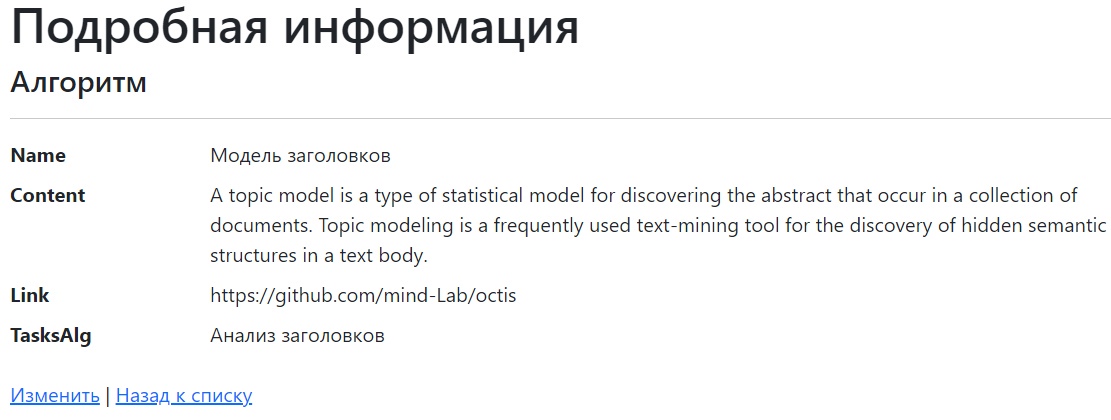


Рисунок 5.6 – Страница с детальной информацией об алгоритме

## 5.6 Страница со списком задач

На данной странице пользователь может ознакомиться со списком задач, имеющих алгоритмы для решения данных задач. Код страницы находится в приложении А. На рисунке 5.7 представлен пример страницы со списком задач.

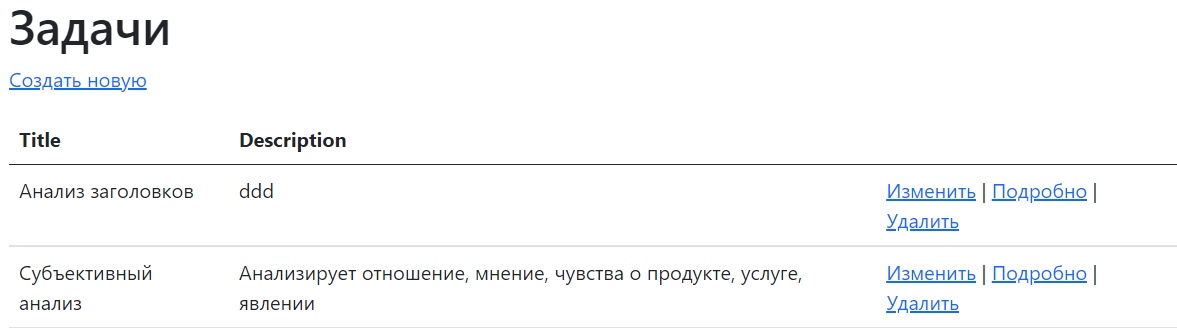


Рисунок 5.7 – Страница со списком задач

## 5.7 Страница с детальной информацией о задаче

На данной странице пользователь может ознакомиться с детальной информацией о задаче и посмотреть список алгоритмов, которые решают данную задачу. При клике по алгоритму происходит переход на страницу выбранного алгоритма. Код страницы представлен в приложении А. На рисунке 5.8 представлен пример подробной информации о задаче.

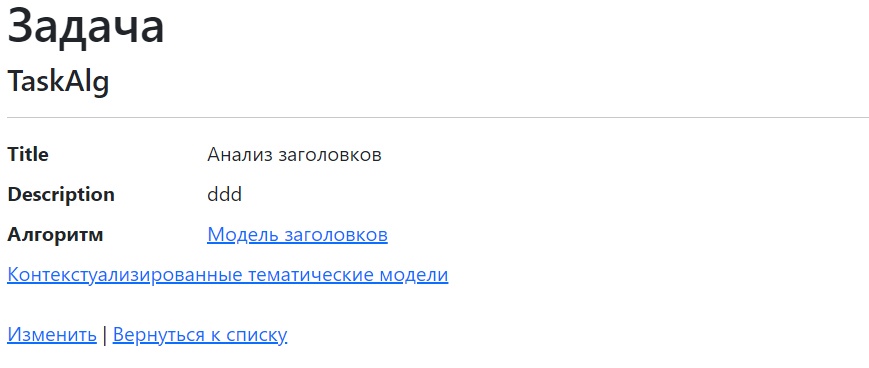


Рисунок 5.8 – Страница с подробной информацией о задаче

ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы был исследован процесс обработки естественного языка, в частности были выявлены основные особенности при работе с технологиями извлечения данных из неструктурированного текста. Были изучены основные задачи, которые решает технология обработки естественного языка и на основе этих данных были определены требования к разрабатываемой системе. Был проведен анализ существующих систем и выявлены их основные преимущества и недостатки. На основе полученных данных была спроектирована база данных для хранения данных об алгоритмах извлечения данных из текста и разработан прототип веб-приложения со следующими функциями:

1. функция авторизации;
2. функция просмотра списка категорий и задач;
3. функция просмотра детальной информации о категориях, задачах, алгоритмах.
4. функция поиска алгоритма по задаче, которую он решает.

Список функций, которые могут быть реализованы при реальной разработке:

1. функция поиска алгоритма по названию;
2. функция сортировки списков;
3. расширить функционал авторизации, разграничив пользователя и администратора.

В ходе разработки прототипа была спроектирована и создана база данных, пользовательский интерфейс и основные функции, необходимые для работы системы, поэтому можно считать, что задание по разработке базы знаний об алгоритмах извлечения данных из текста было выполнено.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. NLP. Основы. Техники. Саморазвитие. Часть 1 / Хабр [Сайт]. (23.01.2019). URL: <https://habr.com/ru/company/abbyy/blog/437008> (дата обращения 23.06.2022).
2. Чудесный мир Word Embeddings: какие они бывают и зачем нужны? / Хабр [Сайт]. (17.07.2017). URL: <https://habr.com/ru/company/ods/blog/329410> (дата обращения 23.06.2022).
3. NLP. Основы. Техники. Саморазвитие. Часть 2: NER / Хабр [Сайт]. (14.05.2019). URL: <https://habr.com/ru/company/abbyy/blog/449514> (дата обращения: 23.06.2022).
4. Марковские случайные поля / Хабр [Сайт]. (23.10.2014). URL: <https://habr.com/ru/post/241317> (дата обращения: 23.06.2022).
5. Bootstrap · The most popular HTML, CSS, and JS library in the world [Сайт]. (01.10.2012). URL: <https://getbootstrap.com/> (дата обращения: 23.06.2022).

ПРИЛОЖЕНИЕ А

(справочное)

Полный код проекта можно найти в репозитории на github: <https://github.com/DeGrimer/NLPDB>

**А.1 Код страницы со списком категорий**

**А.1.1 Код представления**

@model IEnumerable<NLPDB.Web.Data.Category>

@{

ViewData["Title"] = "Index";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h1>Список категорий</h1>

<p>

<a asp-action="Create">Добавить новую категорию</a>

</p>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Name)

</th>

<th>

@Html.DisplayNameFor(model => model.Description)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.Description)

</td>

<td>

<a asp-action="Edit" asp-route-id="@item.Id">Изменить</a> |

<a asp-action="Details" asp-route-id="@item.Id">Подробно</a> |

<a asp-action="Delete" asp-route-id="@item.Id">Удалить</a>

</td>

</tr>

}

</tbody>

</table>

**А.1.2 Код обработчика**

public async Task<IActionResult> Index()

{

return \_context.Categories != null ?

View(await \_context.Categories.ToListAsync()) :

Problem("Entity set 'ApplicationDbContext.Categories' is null.");

}

А.2 Код страницы с детальной информацией о категории

А.2.1 Код представления

@model NLPDB.Web.Data.Category

@{

ViewData["Title"] = "Details";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<div>

<hr />

<dl class="row">

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Name)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Name)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Description)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Description)

</dd>

</dl>

</div>

<div>

<a asp-action="Edit" asp-route-id="@Model?.Id">Изменить</a> |

<a asp-action="Index">Назад к списку</a>

<a asp-controller="Algorithm" asp-action="Index" asp-route-id="@Model?.Id">Показать алгоритмы</a>

</div>

**А.2.2 Код обработчика**

public async Task<IActionResult> Details(Guid? id)

{

if (id == null || \_context.Categories == null)

{

return NotFound();

}

var category = await \_context.Categories

.FirstOrDefaultAsync(m => m.Id == id);

if (category == null)

{

return NotFound();

}

return View(category);

}

**А.3 Код страницы со списком задач**

**А.3.1 Код представления**

@model IEnumerable<NLPDB.Web.Data.TaskAlg>

@{

ViewData["Title"] = "Список задач";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h1>Задачи</h1>

<p>

<a asp-action="Create">Создать новую</a>

</p>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Title)

</th>

<th>

@Html.DisplayNameFor(model => model.Description)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Title)

</td>

<td>

@Html.DisplayFor(modelItem => item.Description)

</td>

<td>

<a asp-action="Edit" asp-route-id="@item.Id">Изменить</a> |

<a asp-action="Details" asp-route-id="@item.Id">Подробно</a> |

<a asp-action="Delete" asp-route-id="@item.Id">Удалить</a>

</td>

</tr>

}

</tbody>

</table>

**А.3.2 Код обработчика**

public async Task<IActionResult> Index()

{

return \_context.TaskAlg != null ?

View(await \_context.TaskAlg.ToListAsync()) :

Problem("Entity set 'ApplicationDbContext.TaskAlg' is null.");

}

**А.4 Код страницы детальной информации о задаче**

**А.4.1 Код представления**

@model NLPDB.Web.Data.TaskAlg

@{

ViewData["Title"] = "Описание задачи";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h1>Задача</h1>

<div>

<h4>TaskAlg</h4>

<hr />

<dl class="row">

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Title)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Title)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Description)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Description)

</dd>

<dt class = "col-sm-2">

Алгоритм

</dt>

@foreach(var t in Model.Algorithms)

{

<dd class = "col-sm-10">

<a asp-controller="Algorithm" asp-action="Details" asp-route-id="@t.Id">@t.Name</a>

</dd>

}

</dl>

</div>

<div>

<a asp-action="Edit" asp-route-id="@Model?.Id">Изменить</a> |

<a asp-action="Index">Вернуться к списку</a>

</div>

**А.4.2 Код обработчика**

public async Task<IActionResult> Details(Guid? id)

{

if (id == null || \_context.TaskAlg == null)

{

return NotFound();

}

var taskAlg = await \_context.TaskAlg

.FirstOrDefaultAsync(m => m.Id == id);

if (taskAlg == null)

{

return NotFound();

}

var algs = await \_context.Algorithms.Where(x => x.TasksAlg.Contains(taskAlg)).ToListAsync();

if(algs != null)

{

taskAlg.Algorithms = algs;

}

return View(taskAlg);

}

**А.5 Код страницы со списком алгоритмов**

**А.5.1 Код представления**

@model IEnumerable<NLPDB.Web.Data.Algorithm>

@{

}

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Name)

</th>

<th>

@Html.DisplayNameFor(model => model.TasksAlg)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.TasksAlg)

</td>

<td>

<a asp-action="Details" asp-route-id="@item.Id">Подробно</a>

</td>

</tr>

}

</tbody>

</table>

**А.5.2 Код обработчика**

public async Task<IActionResult> Index(Guid? id)

{

var alg1 = \_context.Categories.Where(x => x.Id == id).First();

var alg2 = \_context.Algorithms.Where(x => x.Category == alg1).ToList();

return View(alg2);

}

**А.6 Код страницы с детальной информацией об алгоритме**

**А.6.1 Код представления**

@model NLPDB.Web.Data.Algorithm

@{

ViewData["Title"] = "Details";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h1>Подробная информация</h1>

<div>

<h4>Алгоритм</h4>

<hr />

<dl class="row">

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Name)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Name)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Content)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Content)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Link)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Link)

</dd>

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.TasksAlg)

</dt>

@foreach(var t in Model.TasksAlg)

{

<dd class = "col-sm-10">

@t.Title

</dd>

}

</dl>

</div>

<div>

<a asp-action="Edit" asp-route-id="@Model?.Id">Изменить</a> |

<a asp-action="Index" asp-route-id="@Model?.Category.Id">Назад к списку</a>

</div>

**А.6.2 Код обработчика**

public async Task<IActionResult> Details(Guid? id)

{

if (id == null || \_context.Algorithms == null)

{

return NotFound();

}

var algorithm = await \_context.Algorithms

.FirstAsync(m => m.Id == id);

var category = await \_context.Categories

.FirstOrDefaultAsync(c => c.Algorithms.Contains(algorithm));

algorithm.Category = category;

if (algorithm == null)

{

return NotFound();

}

var tasks = await \_context.TaskAlg.Where(x => x.Algorithms.Contains(algorithm)).ToListAsync();

algorithm.TasksAlg = tasks;

return View(algorithm);

}

ПРИЛОЖЕНИЕ Б

(справочное)

**Б.1 Код для инициализации базы данных с начальными тестовыми данными.**

public static class DataInitializer

{

public static async Task InitializeAsync(IServiceProvider serviceProvider)

{

var scope = serviceProvider.CreateScope();

await using var context = scope.ServiceProvider.GetService<ApplicationDbContext>();

var isExists = context!.Database.CanConnect();

if (isExists)

{

return;

}

await context.Database.MigrateAsync();

var roles = AppData.Roles.ToArray();

var roleStore = new RoleStore<IdentityRole>(context);

foreach (var role in roles)

{

if (!context.Roles.Any(x => x.Name == role))

{

await roleStore.CreateAsync(new IdentityRole(role) { NormalizedName = role.ToUpper() });

}

}

const string username = "d@f.com";

if (context.Users.Any(x => x.Email == username))

{

return;

}

var user = new IdentityUser

{

Email = username,

EmailConfirmed = true,

NormalizedEmail = username.ToUpper(),

PhoneNumber = "+7900800553535",

UserName = username,

NormalizedUserName = username.ToUpper(),

SecurityStamp = Guid.NewGuid().ToString("D")

};

var passwordHasher = new PasswordHasher<IdentityUser>();

user.PasswordHash = passwordHasher.HashPassword(user, "123456");

var userStore = new UserStore<IdentityUser>(context);

var identityResult = await userStore.CreateAsync(user);

if (!identityResult.Succeeded)

{

throw new ArgumentException();

}

var userManger = scope.ServiceProvider.GetService<UserManager<IdentityUser>>();

foreach (var role in roles)

{

var identityResultRole = await userManger!.AddToRoleAsync(user, role);

if (!identityResultRole.Succeeded)

{

throw new ArgumentException();

}

}

var category = new Category()

{

Name = "Классификация текста",

Description = "Text classification is the task of assigning a sentence or document an appropriate category. The categories depend on the chosen dataset and can range from topics."

};

await context.Categories.AddAsync(category);

category = new Category()

{

Name = "Анализ тональности",

Description = "Sentiment analysis is the task of classifying the polarity of a given text. For instance, a text-based tweet can be categorized into either \"positive\", \"negative\", or \"neutral\". ",

};

await context.Categories.AddAsync(category);

await context.SaveChangesAsync();

var algorithm = new Algorithm()

{

Category = context.Categories.Where(c => c.Name == "Классификация текста").First(),

Name = "Модель заголовков",

Content = "A topic model is a type of statistical model for discovering the abstract that occur in a collection of documents. Topic modeling is a frequently used text-mining tool for the discovery of hidden semantic structures in a text body.",

Link = "https://github.com/mind-Lab/octis"

};

await context.Algorithms.AddAsync(algorithm);

algorithm = new Algorithm()

{

Category = context.Categories.Where(c => c.Name == "Анализ тональности").First(),

Name = "Анализ тональности твиттера",

Content = "Twitter sentiment analysis is the task of performing sentiment analysis on tweets from Twitter.",

Link = "https://github.com/lopezbec/COVID19\_Tweets\_Dataset"

};

await context.Algorithms.AddAsync(algorithm);

algorithm = new Algorithm()

{

Category = context.Categories.Where(c => c.Name == "Классификация текста").First(),

Name = "Двойное контрастное обучение",

Content = "Contrastive learning has achieved remarkable success in representation learning via self-supervision in unsupervised settings. However, effectively adapting contrastive learning to supervised learning tasks remains as a challenge in practice.",

Link = "https://github.com/hiyouga/dual-contrastive-learning"

};

await context.Algorithms.AddAsync(algorithm);

algorithm = new Algorithm()

{

Category = context.Categories.Where(c => c.Name == "Классификация текста").First(),

Name = "Контекстуализированные тематические модели",

Content = "Contextualized Topic Models are based on the Neural-ProdLDA variational autoencoding approach by Srivastava and Sutton (2017).",

Link = "https://github.com/MilaNLProc/contextualized-topic-models"

};

await context.Algorithms.AddAsync(algorithm);

await context.SaveChangesAsync();

var tag = new TaskAlg()

{

Title = "Анализ заголовков",

Description = "ddd",

Algorithms = new List<Algorithm>()

{

context.Algorithms.Where(t => t.Name == "Модель заголовков").First(),

context.Algorithms.Where(t => t.Name == "Контекстуализированные тематические модели").First()

}

};

await context.TaskAlg.AddAsync(tag);

tag = new TaskAlg()

{

Title = "Субъективный анализ",

Description = "Анализирует отношение, мнение, чувства о продукте, услуге, явлении",

Algorithms = new List<Algorithm>()

{

context.Algorithms.Where(t => t.Name == "Двойное контрастное обучение").First(),

context.Algorithms.Where(t => t.Name == "Анализ тональности твиттера").First()

}

};

await context.TaskAlg.AddAsync(tag);

await context.SaveChangesAsync();

}

}

**Б.2 Код для создания миграций для базы данных**

public partial class AddEntity : Migration

{

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.CreateTable(

name: "Category",

columns: table => new

{

Id = table.Column<Guid>(type: "uniqueidentifier", nullable: false),

Name = table.Column<string>(type: "nvarchar(100)", maxLength: 100, nullable: false),

Description = table.Column<string>(type: "nvarchar(3000)", maxLength: 3000, nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Category", x => x.Id);

});

migrationBuilder.CreateTable(

name: "TaskAlg",

columns: table => new

{

Id = table.Column<Guid>(type: "uniqueidentifier", nullable: false),

Title = table.Column<string>(type: "nvarchar(100)", maxLength: 100, nullable: false),

Description = table.Column<string>(type: "nvarchar(2000)", maxLength: 2000, nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_TaskAlg", x => x.Id);

});

migrationBuilder.CreateTable(

name: "Algorithm",

columns: table => new

{

Id = table.Column<Guid>(type: "uniqueidentifier", nullable: false),

Name = table.Column<string>(type: "nvarchar(150)", maxLength: 150, nullable: false),

Content = table.Column<string>(type: "nvarchar(3000)", maxLength: 3000, nullable: false),

Link = table.Column<string>(type: "nvarchar(max)", nullable: false),

CategoryId = table.Column<Guid>(type: "uniqueidentifier", nullable: false),

CreatedAt = table.Column<DateTime>(type: "datetime2", nullable: false),

CreatedBy = table.Column<string>(type: "nvarchar(max)", nullable: true),

UpdatedAt = table.Column<DateTime>(type: "datetime2", nullable: true),

UpdatedBy = table.Column<string>(type: "nvarchar(max)", nullable: true)

},

constraints: table =>

{

table.PrimaryKey("PK\_Algorithm", x => x.Id);

table.ForeignKey(

name: "FK\_Algorithm\_Category\_CategoryId",

column: x => x.CategoryId,

principalTable: "Category",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "CategoryTaskAlg",

columns: table => new

{

CategoriesId = table.Column<Guid>(type: "uniqueidentifier", nullable: false),

TasksAlgId = table.Column<Guid>(type: "uniqueidentifier", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_CategoryTaskAlg", x => new { x.CategoriesId, x.TasksAlgId });

table.ForeignKey(

name: "FK\_CategoryTaskAlg\_Category\_CategoriesId",

column: x => x.CategoriesId,

principalTable: "Category",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_CategoryTaskAlg\_TaskAlg\_TasksAlgId",

column: x => x.TasksAlgId,

principalTable: "TaskAlg",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "AlgorithmTaskAlg",

columns: table => new

{

AlgorithmsId = table.Column<Guid>(type: "uniqueidentifier", nullable: false),

TasksAlgId = table.Column<Guid>(type: "uniqueidentifier", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_AlgorithmTaskAlg", x => new { x.AlgorithmsId, x.TasksAlgId });

table.ForeignKey(

name: "FK\_AlgorithmTaskAlg\_Algorithm\_AlgorithmsId",

column: x => x.AlgorithmsId,

principalTable: "Algorithm",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_AlgorithmTaskAlg\_TaskAlg\_TasksAlgId",

column: x => x.TasksAlgId,

principalTable: "TaskAlg",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateIndex(

name: "IX\_Algorithm\_CategoryId",

table: "Algorithm",

column: "CategoryId");

migrationBuilder.CreateIndex(

name: "IX\_AlgorithmTaskAlg\_TasksAlgId",

table: "AlgorithmTaskAlg",

column: "TasksAlgId");

migrationBuilder.CreateIndex(

name: "IX\_CategoryTaskAlg\_TasksAlgId",

table: "CategoryTaskAlg",

column: "TasksAlgId");

}

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropTable(

name: "AlgorithmTaskAlg");

migrationBuilder.DropTable(

name: "CategoryTaskAlg");

migrationBuilder.DropTable(

name: "Algorithm");

migrationBuilder.DropTable(

name: "TaskAlg");

migrationBuilder.DropTable(

name: "Category");

}

}