МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
имени первого Президента России Б. Н. Ельцина

ИНСТИТУТ ЕСТЕСТВЕННЫХ НАУК И МАТЕМАТИКИ

Кафедра вычислительной математики и компьютерных наук

СОЗДАНИЕ ЦИФРОВОЙ ПЛАТФОРМЫ ФОЛЬКЛОРНОГО АРХИВА УРФУ

Направление 09.04.03 «Прикладная информатика»

Допустить к защите:	Выпускная квалификационная работа
Директор департамента:	**Шмонина Ирина Викторовна**
_____	_____
Нормоконтролер:	Научный руководитель: к. ф-м. н. доцент КВМиКН, А.Ю. Коврижных
_____	_____

Екатеринбург
2020

РЕФЕРАТ

Выпускная квалификационная работа Шмониной Ирины Викторовны на тему: «СОЗДАНИЕ ЦИФРОВОЙ ПЛАТФОРМЫ ФОЛЬКЛОРНОГО АРХИВА УРФУ»: стр. 31, рис. 3, таб. 1.

Ключевые слова: web-сайт, системный анализ, проектирование базы данных, проектирование REST API, ASP.NET Core, React

В данной работе рассмотрены современные методики и инструменты создание сервисов от анализа потребностей клиента, до реализации клиентской части сервиса.

A thesis research of Shmonina Irina Victorovna on a topic: "CREATION OF THE DIGITAL PLATFORM FOR THE FOLKLORE ARCHIVE OF UrFU". The thesis research set out on 31 pages, including 3 figures and 1 table.

Key words: website, system analysis, database design, REST API design, ASP.NET Core, React.

In this research, modern methods and tools for creating services from analysis of client needs to the implementation of the client part of the service are considered.

МЕСТО ВЫПОЛНЕНИЯ РАБОТЫ

Место выполнения работы Уральский Федеральный Университет имени первого Президента России Б.Н. Ельцина, институт естественных наук и математики, кафедра вычислительной математики и компьютерных наук.

СОДЕРЖАНИЕ

Оглавление

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

CMS система — Content Management System это система управления контентом, набор скриптов для создания, редактирования и управления контентом сайта

БД — база данных

MVP — минимально жизнеспособный продукт или minimum viable product

API — Application programming interface

ВВЕДЕНИЕ

Фольклорный архив УрФУ содержит более 250 000 фольклорных текстов. Этот материал собирался на протяжении 60 лет работы фольклорной экспедиции УрГУ/УрФУ. В настоящие время архив существует только в рукописном виде. Это одна из крупнейших региональных фольклорных коллекций в стране.

В настоящий момент в России не существует подобных электронных ресурсов, многие фольклорные материалы не опубликованы, хранятся в университетских депозитариях в бумажном виде и соответственно остаются труднодоступными для исследователей.

Для обеспечения компактного хранения больших объемов документов и организации системы их быстрого поиска нужно создать электронный архив. Данная система позволит массово оцифровать бумажные документы, систематизировать их и надежно хранить большой объём электронных документов. Поисковые возможности, которые могут быть реализованы при создании электронного корпуса русского фольклора, в перспективе позволит совершенно по-новому представить картину бытования фольклора во времени и пространстве, в различных социальных средах, а также по-новому взглянуть на историю полевой фольклористики в нашей стране.

Оцифровка и научная онлайн публикация фольклора, как уже опубликованного в научных изданиях, так и существующего лишь в виде карточек полевых экспедиций, может стать важным имиджевым проектом для УрФУ, поскольку неизбежно привлечет к себе внимание исследователей как в России, так и за рубежом. Масштабирование проекта существенно повышает его национально-культурное и научное значение, а значит, позволит привлекать регулярное грантовое финансирование. Кроме того, проектируемая CMS система в дальнейшем может быть адаптирована для создания аналогичных коллекций литературных и языковых материалов, коллекций исторических источников и так далее.

ОСНОВНАЯ ЧАСТЬ

1. Постановка задачи работы

В рамках проекта требуется:

а) Оцифровать тексты, создать электронные копии текстовых материалов

б) Провести аналитику и выявить основные требования к проекту, выяснить проблемы и сценарии

в) Спроектировать электронную базу данных с указанием для каждого текста атрибутов и классификационных признаков, которые могут быть использованы для работы с БД

г) Спроектировать API архива для работы с документами

д) Разработать электронную онлайн платформу, позволяющую, с одной стороны, удаленно пополнять электронную коллекцию фольклорных текстов, а с другой стороны формировать выборки по заданным параметрам, осуществлять полноценный поиск

е) Сформировать MVP и разработать ядро CMS и заполнить его уже оцифрованными материалами фольклорного архива УрФУ

2. Раздел

2.1. Способы и методы теоретических и аналитических исследований

2.1.1. Исследование потребностей

Для формирования качественного и нужного продукта нужно провести исследование потребностей клиента. Одна из методик проведения исследования – глубинное интервью.

Глубинное интервью – это беседа с респондентом в форме, побуждающей последнего к подробным ответам на заданную тему. Позволяет выявить сценарии, скрытые мотивы, ценности, убеждения, проблемы и потребности респондента.

Интервью включает в себя:

- Формулировку целей исследования

- Составление списка вопросов

- Проведение интервью

- Обработка результатов

Цель исследования:

Формирование ценности продукта, исследование сценариев пользователей, выделение активных ролей, формирование основной функциональности, формирование дополнительной функциональности.

Результаты исследования представлены в таблицу 2.1.1.1.

Таблица 2.1.1.1 – Результаты глубинного интервью

№	Вопрос	Ответ
1.	Для чего нужен проект? В чем ценность?	Для формирования электронного архива. Ценность: быстрый доступ

2.	Какой формат документов?	В основном это рукописные записи в очень плохом состоянии. Предполагается, что хранить будут сканы и их расшифровки
3.	Какую дополнительную информацию нужно хранить у документа?	Жанр, место, время текста, информант, фольклорист, мотивно-тематическая классификация, ключевые слова, источник, библиография о тексте, морфологический анализ
4.	Кто пользуется системой?	Фольклористы – исследователи
5.	Как каждая роль использует систему? Какую функцию выполняет?	Есть обычные пользователи. Они используют систему для поиска нужных документов. Поиск происходит по дополнительной информации. Есть администраторы, которые могут добавлять или удалять документы, редактировать информацию
6.	Без чего на первом этапе проект не может существовать?	Без добавления / удаления / редактирования документов
7.	Как в дальнейшем хочется развивать проект?	Составить карту фольклорного быта
8.	Где разворачивается проект?	На серверах УрФУ

2.1.2. Roadmap

Для успешного запуска проекта требуется составить план. Основа плана — стратегия, а для ее реализации требуется вспомогательное средство. Это roadmap или дорожная карта. Инструмент, который используется в разработке сложных технических продуктов. Рассмотрим подробнее на рисунке 2.1.2.1.

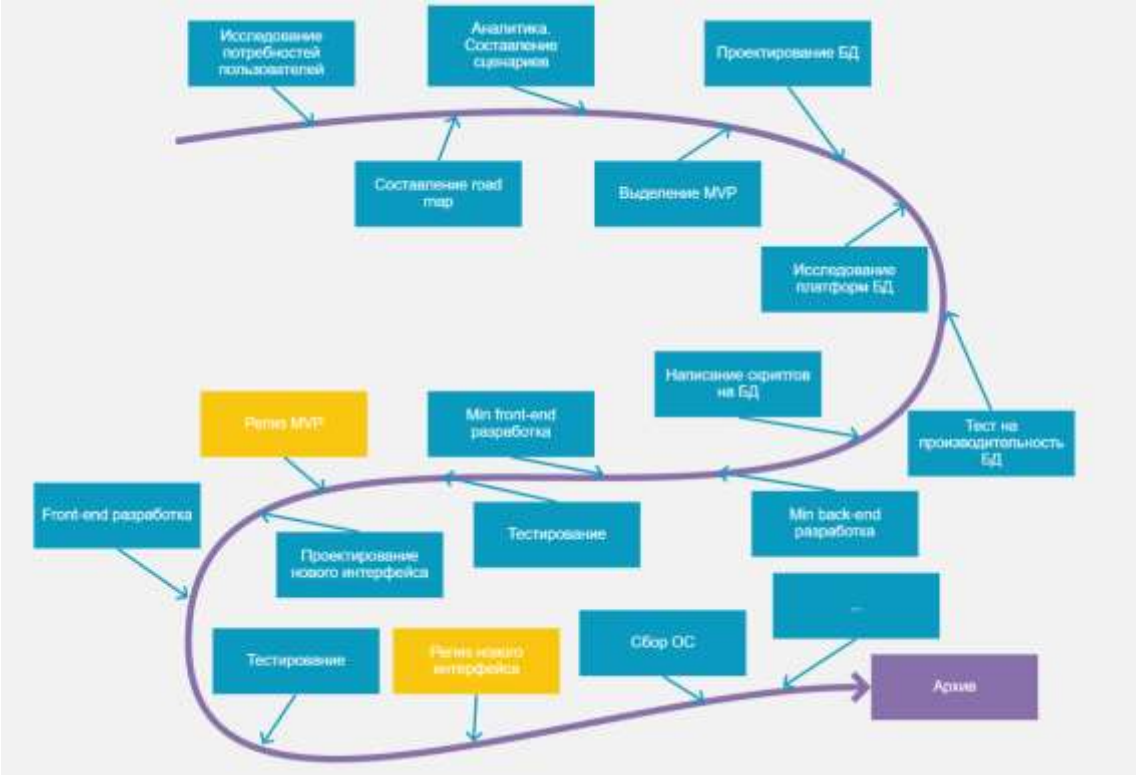

Рисунок 2.1.2.1 – Дорожная карта проекта

Дорожная карта является полезным инструментом для управления ожиданиями заинтересованных сторон, а также для планирования ресурсов.

Этапы создания продукта:

а) Исследование потребностей. В этап входит исследование проблем и предпосылок возникновения проекта

б) Составление roadmap.

в) Аналитика, составление сценариев

г) Выделение MVP

д) Проектирование БД

е) Написание скриптов для БД

ж) Реализация минимального back-end

з) Проектирование API

и) Реализация минимального front-end

к) Тестирование

л) Релиз

2.1.3. Исследование конкурентов

В разборе сценариев, портретов клиентов, потребностей пользователей очень помогает исследование конкурентов и поиск аналогичных систем.

Рассмотрим несколько похожих систем:

а) Национальный корпус русского языка - http://www.ruscorpora.ru.

На этом сайте помещен корпус современного русского языка общим объемом более 600 млн слов. Корпус русского языка — это информационно-справочная система, основанная на собрании русских текстов в электронной форме. Отличительной чертой сайта является поиск текстов. На сайте предусмотрены различные комбинации фильтров. Есть сложный поиск, есть поиск по регулярным выражениям. Есть также поиск по различным критериям, таких как жанр, грамматический признак и так далее. На сайте предусмотрена возможность поиска текста по n-граммам. Также на сайте представлен морфологический анализ каждого слова из текста.

При грамматической разметке текстов (морфологической и семантической) на сайте используются программы морфологического анализа Mystem (разработка компании Яндекс) и Dialing (коллектив авторов под руководством А. В. Сокирко). Эти программы Mystem и Dialing распространяются свободно и доступны для некоммерческого использования согласно соответстветствующим лицензионным соглашениям. В сервисе фольклорного архива также можно использовать данные программы.

б) Национальная электронная библиотека - https://rusneb.ru/

На данный момент в НЭБ доступно 1 671 878 электронных книг и 29 948 245 записей каталогов. Это древние рукописи, советские журналы, учебники и так далее. Оцифрованные документы можно скачать или полистать прямо в браузере. В разделе «Коллекции» составляются подборки по авторам и событиям. Сложных фильтров на сайте нет, как и морфологического разбора.

2.1.4. Сценарии

В нашей системе есть 2 вида пользователей: администратор и читатель.

Функциональность для читателя:

а) Найти книги по:
   1) жанру
   2) месту
   3) времени текста
   4) информанту
   5) фольклористу
   6) мотивно-тематической классификации
   7) ключевым словам
   8) источнику
   9) библиографии о тексте
   10) любой комбинации из выше стоящих элементов

б) Открыть карточку документа или сам документ

в) Скачать документ

г) Зарегистрироваться в системе и получить доступ к функциональности администратора

Функциональность администратора:

а) Найти книги по:
   1) жанру
   2) месту
   3) времени текста
   4) информанту
   5) фольклористу
   6) мотивно-тематической классификации
   7) ключевым словам
   8) источнику

9) библиографии о тексте

10) любой комбинации из выше стоящих элементов

б) Авторизоваться

в) Изменить параметры у документа

1) жанр

2) место

3) время текста

4) информант

5) фольклорист

6) мотивно-тематическая классификация

7) ключевые слова

8) источник

9) библиография о тексте

г) Изменить сам документ

д) Удалить документ

е) Удалить дополнительную информацию у самого документа

ж) Добавлять других администраторов

2.1.5. MVP

Минимально жизнеспособный продукт или minimum viable product — продукт, обладающий минимальными, но достаточными для удовлетворения первых потребителей функциями. Основная задача создания MVP — получение обратной связи для формирования гипотез дальнейшего развития продукта.

В нашем сервисе мы предусматриваем три раздела: страница о проекте, список всех документов с возможностью фильтрации и постраничным просмотром документов, также требуется раздел для создания документов.

В нашем сервисе также нужно предусмотреть форму для документа. В карточке документа должна быть возможность указать название документа, указать расшифровку документа, указать год записи, указать информантов, указать фольклористов, указать жанры, мотив или тематику, указать ключевые слова, указать дополнительную информацию, указать место сбора документа, а также произвести морфологический анализ текста. Также должна быть возможность редактировать, сохранить, удалить и восстановить документ.

2.1.6. Проектирование базы данных

Проектирования БД – переход от неформального словесного описания информационной структуры предметной области к формализованному описанию объектов предметной области в терминах некоторой модели. Было проведены следующие этапы:

а) Системный анализ и словесное описание информационных объектов предметной области
б) Проектирование базы данных в терминах модели данных
в) Физическое проектирование БД, развертывание в сервисе Azure

Рассмотрим подробнее первый этап – системный анализ и сбор требований. В нашей Базе данных требовалось хранить такие сущности как документ, фольклорист, информант, жанр, классификатор, тег. Рассмотрим подробнее каждую сущность:

Документ – сущность для хранения данных о документе. Например:

а) Идентификатор (Id) – уникальный ключ для документа
б) Название документа (Title) – предполагается, что это будет строка, в которой будет в краткой форме записано название
в) Расшифровка документа (Content) – так как многие тексты в архиве в трудночитаемом состоянии, то многие тексты были переведены в электронный вид. Теперь нудно хранить расшифровку текстов в сервисе
г) Место (PlaceName, PlaceLatitude, PlaceLongitude) – документ имеет принадлежность к месту, где его собрали. Для визуализации текстов на карте требуется кроме названия места хранить долготу и широту
д) Год записи (YearOfRecord) – год, в котором документ был записан в архив
е) Дополнительная информация (AdditionalInformation) – комментарий к документу

ж) Файл (FileName, FileId) – в нашем сервисе будет возможность загрузить скан документа в сервис, для этого в сущности документ требуется 2 поля – наименование загруженного файла и идентификатор этого файла для хранения документа на сервере

з) Дополнительные поля для работы сервиса (Deleted, CreatedAt)

1) Deleted – признак того, что документ был удален. Нужен для показа актуальной информации
2) CreatedAt – дата создания документа. Нужен для сортировки документов

Фольклорист – человек, который сделал запись о документе. В сущность фольклорист нужно указывать следующие параметры:

а) Идентификатор (Id) – идентификатор уникальной записи в таблицу
б) Фамилия Имя и Отчество фольклориста (FIO) – строка для записи ФИО сотрудника

Информант – человек, который рассказал историю, частушку или другой жанр произведения.

Сущность Информант должна содержать следующие поля:

а) Идентификатор (Id) – идентификатор уникальной записи в таблицу
б) Фамилия Имя и Отчество информанта (FIO) – строка для записи ФИО человека, который предоставил материал
в) Год рождения (YearOfBirth) – год рождения информанта. Нужно для понимания, какие особенности речи характерны для людей определенной эпохи

Жанр - совокупность формальных и содержательных особенностей произведения.

Сущность Жанр должна содержать следующие поля:

а) Идентификатор (Id) – идентификатор уникальной записи в таблицу

б) Название жанра (GenreName) – строка, в которой будет хранится название жанра

Мотивно-тематическая классификация. Сущность Класификатора должна содержать следующие поля:

а) Идентификатор (Id) – идентификатор уникальной записи в таблицу
б) Код классификации (Code) – строка, в которой будет хранится код
в) Наименование классификации (ClassificationName) – название классификации

Сущность Теги должна содержать следующие поля:

а) Идентификатор (Id) – идентификатор уникальной записи в таблицу
б) Наименование тега (TagName)

Промежуточные таблицы

Для осуществления связей многие ко многим требуется создать промежуточные таблицы для следующих сущностей:

1. Документы – Информанты
2. Документы – Фольклористы
3. Документы – Жанры
4. Документы – Классификации
5. Документы - Теги

Также есть отдельная таблица для хранения морфологического анализа.

Результат проектирования базы данных представлен на рисунке:

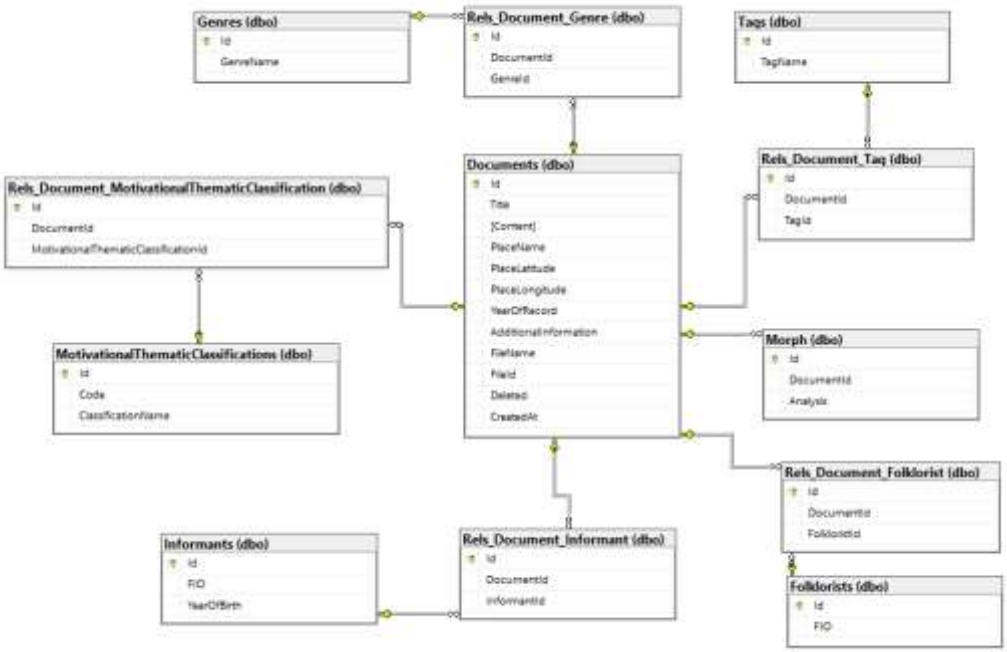

Рисунок 2.1.6.1 – Спроектированная база данных для электронного архива фольклорных документов

2.1.7. Проектирование API

Большинство современных веб-приложений предоставляют API, которые клиенты могут использовать для взаимодействия с приложением. Для того, чтобы спроектировать API нужно перечислить все типы данных, которые клиентское приложение может захотеть получить или передать с помощью сервиса. Такое описание называется семантическим, так как отображает значение данных в приложении, и описание того, что происходит в самом приложении.

Сущность «Документ» имеет следующие семантические описания: идентификатор (Id тип int) – уникальный ключ для документа, название документа (Title тип string), расшифровка документа (Content тип string), название места сбора информации и координаты места (PlaceName тип string, PlaceLatitude тип double, PlaceLongitude тип double), год записи документа (YearOfRecord тип int), дополнительная информация (AdditionalInformation тип string), отсканированный файл (FileName тип string, FileId тип string), список жанров (Genres массив типа Genre), список информантов (Informants массив типа Informant), список фольклористов (Folklorists массив типа Folklorist), список мотивов (MotivationalThematicClassifications массив типа MotivationalThematicClassification), список тегов (Tags массив типа Tag) и морфологический разбор текста (Morph тип string).

Сущность «Жанр» имеет следующие поля: идентификатор (Id тип int) – уникальный ключ для жанра, название жанра (GenreName тип string).

У сущности «Информант» есть следующие поля: идентификатор (Id тип int) – уникальный ключ для информанта, фамилия имя и отчество информанта (FIO тип string) и год рождения (YearOfBirth тип int).

У сущности «Фольклорист» есть следующие поля: идентификатор (Id тип int) – уникальный ключ для фольклориста, фамилия имя и отчество фольклориста (FIO тип string).

Сущность «Мотивно-тематический классификатор» имеет следующие поля: идентификатор (Id тип int) – уникальный ключ для классификатора, название мотива или тематики (ClassificationName тип string) и код (Code тип string)

Сущность «Тег» имеет идентификатор (Id тип int) – уникальный ключ для тега, название тега (TagName тип string).

Следующим этапом спроектируем возможные состояния в сервисе. Для этого используем диаграмму состояний для предполагаемого API. Каждый блок на диаграмме представляет возможное состояние, стрелки используются для обозначения переходов от одного блока к другому, от одного состояния к следующему. Эти переходы инициируются запросами.

Приложение для хранения фольклорных документов может потребовать доступ к карточке документа, возможно фильтровать документы, можно создавать документы, можно искать, создавать и обновлять такие сущности как фольклорист, информант, жанр, мотивно-тематический классификатор, тег. Многие из этих действий используют значения состояний для передачи данных между клиентом и сервером. Например, действие «добавить документ» позволяет клиенту передавать значения состояний Title, Content, PlaceName, PlaceLatitude, PlaceLongitude, YearOfRecord, AdditionalInformation, FileName, FileId, Genres, Informants, Folklorists, MotivationalThematicClassifications, Tags и Morph. Ниже на рисунке 2.1.7.1 показаны основные действия по документам.

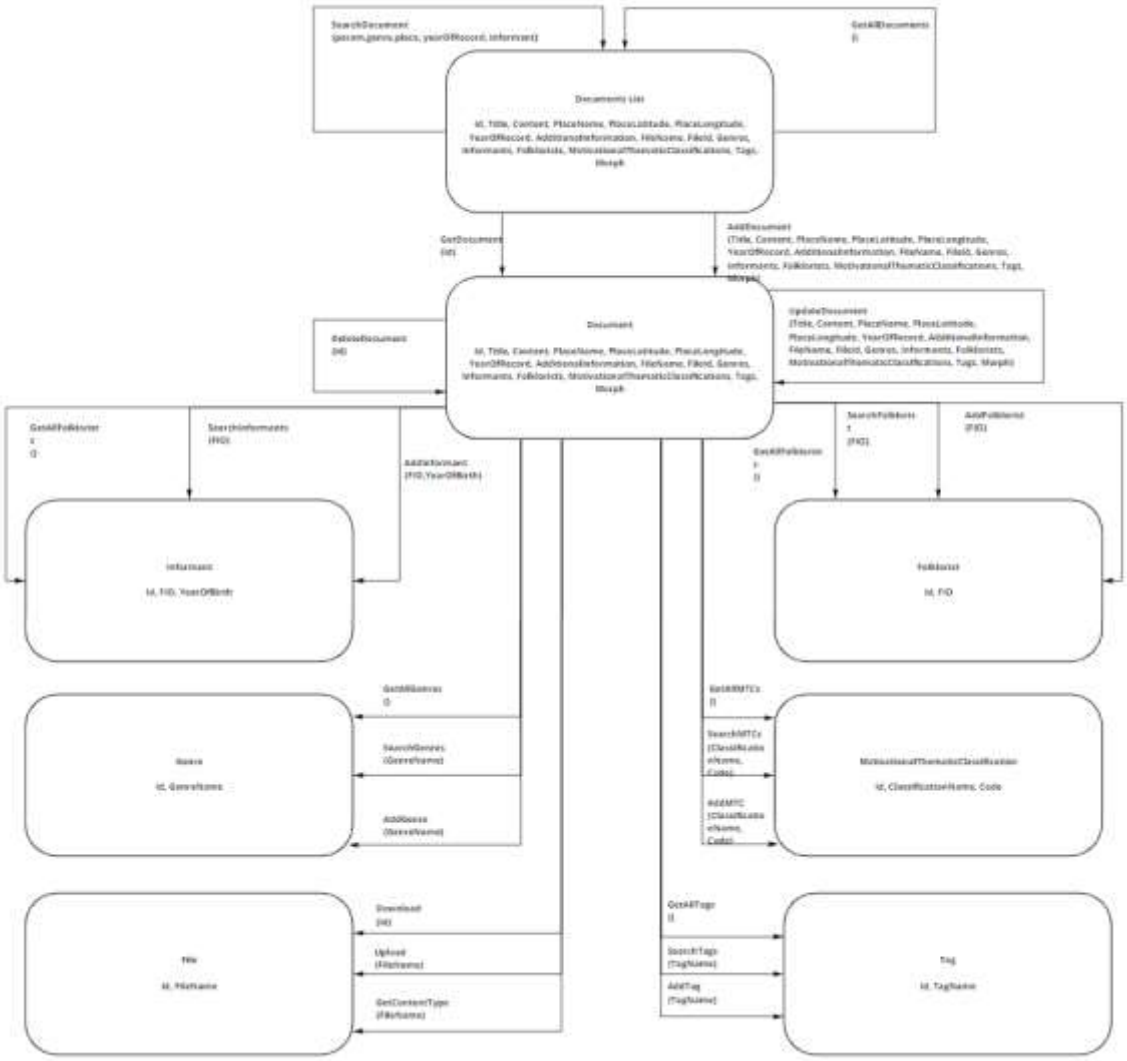

Рисунок 2.1.7.1 – Проектирование REST API

2.2. Способы и методы решения задачи

2.2.1. Примененные инструменты и технологии

Для разработки базы данных я использовала MSSql. SQL Server является одной из наиболее популярных систем управления базами данных (СУБД) в мире. Данная СУБД подходит для самых различных проектов: от небольших приложений до больших высоконагруженных проектов. Для организации баз данных MS SQL Server использует реляционную модель. Реляционная модель предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта.

В MS SQL Server требовалось создать 12 таблиц: документы, информанты, фольклористы, жанры, мотивно-тематический классификатор, теги, морфологический разбор, а также таблицы связи между документом и информантом, документом и фольклористом, документов и жанром, документом и мотивно-тематическим классификатором, документом и тегами. Для этого воспользуемся скриптами для создания таблиц. PRIMARY KEY – указывает на первичный ключ, который однозначно идентифицирует каждую строку/запись в таблице базы данных. Первичные ключи должны содержать уникальные значения. Первичный ключ столбец не может иметь значения NULL. FOREIGN KEY используется для ограничения по ссылкам. Все значения в одном поле таблицы представлены в поле другой таблицы, говорится, что первое поле ссылается на второе. Это указывает на прямую связь между значениями двух полей. FOREIGN KEY используется в таблицах связи между документом и информантом, документом и фольклористом, документов и жанром, документом и мотивно-тематическим классификатором, документом и тегами.

Скрипты создания таблиц представлены в приложении 1.

Также требовалось создать скрипты для наполнения всех таблиц.

Скрипты наполнения таблиц представлены в приложении 1.

В сервисе будет возможность искать документы по заданным параметрам, таким как текст содержит, жанр, информант, год записи, место сбора и так далее. Поэтому нужны были еще скрипты на поиск. Для того, чтобы собрать всю информацию по документу из разных таблиц нужно либо использовать команду JOIN, либо использовать IN в теле запроса.

Рассмотрим каждый из вариантов. Начнем с JOIN. JOIN — оператор языка SQL, который является реализацией операции соединения реляционной алгебры. У JOIN есть четыре реализации: inner, left, right, full. Так как основным элементом является документ, а в смежных таблицах может отсутствовать запись, то для построения всей информации по документу будем использовать LEFT JOIN и присоединять данные к документу.

Пример того, как будет выглядеть скрипт на поиск документов, например, по жанру.

```
SELECT d.*
FROM [dbo].[Documents] d
LEFT JOIN [dbo].[Rels_Document_Folklorist] rdf ON d.Id=rdf.DocumentId
LEFT JOIN [dbo].[Folklorists] f ON rdf.FolkloristId=f.Id
LEFT JOIN [dbo].[Rels_Document_Informant] rdi ON d.Id=rdi.DocumentId
LEFT JOIN [dbo].[Informants] i ON rdi.InformantId=i.Id
LEFT JOIN [dbo].[Rels_Document_Genre] rdg ON d.Id=rdg.DocumentId
LEFT JOIN [dbo].[Genres] g ON g.Id=rdg.GenreId
LEFT JOIN [dbo].[Rels_Document_MotivationalThematicClassification] rdmtc ON d.Id=rdmtc.DocumentId
```

LEFT JOIN [dbo].[MotivationalThematicClassifications] mtc ON mtc.Id=rdmtc.MotivationalThematicClassificationId

LEFT JOIN [dbo].[Rels_Document_Tag] rdt ON d.Id=rdt.Id

LEFT JOIN [dbo].[Tags] t ON t.Id=rdt.TagId

LEFT JOIN [dbo].[Morph] m ON d.Id=m.DocumentId

WHERE g.GenreName LIKE N'%Предание%'

Время выполнения такого запроса на чуть больше 10 000 документах – 2 секунды.

Из минусов такого подхода – количество JOIN и количество выведенных строк.

Если у документа 2 информанта, то мы получим 2 записи в таблице. Если еще 2 фольклориста, то уже 4 записи. Если 2 жанра, то уже 8 записей. Можно заметить, что количество записей растет экспоненциально. В дальнейшем будет сложнее обработать такой результат в ядре приложения.

Рассмотрим следующий подход – запрос через IN. Оператор SQL IN позволяет определить, совпадает ли значение объекта со значением в списке.

Пример того, как будет выглядеть скрипт на поиск документов через IN, например, по жанру.

SELECT d.*

FROM [dbo].[Documents] d

WHERE d.Id IN (SELECT[DocumentId] FROM[dbo].[Rels_Document_Genre] WHERE[GenreId] IN

(SELECT[Id] FROM[dbo].[Genres] WHERE[GenreName] LIKE N'%Предание%'))

Время выполнения такого запроса на чуть больше 10 000 документах – 1 секунды.

Но при такой реализации есть возможность искать документы по заданным параметров без операции JOIN со смежными таблицами.

Для того, чтобы развернуть приложение на сервере использовала Azure. Это облачная платформа Microsoft. Предоставляет возможность разработки и выполнения приложений и хранения данных на серверах.

Для проработки ядра использовала платформу ASP.NET Core. Она предназначена для создания различного рода веб-приложений: от небольших веб-сайтов до крупных веб-порталов и веб-сервисов. ASP.NET Core является opensource-фреймворком. Благодаря модульности фреймворка все необходимые компоненты веб-приложения могут загружаться как отдельные модули через пакетный менеджер Nuget.

Для работы с sql-запросами я использовала Dapper. Он представляет собой технологию сопоставления результатов sql-запросов с классами C#. В этом плане Dapper похож на Entity Framework. В то же время за счет своей легковесности Dapper обеспечивает большую производительность и быстрее позволяет выполнять запросы, нежели Entity Framework. Также в Dapper вы сами формируете запрос, который вы отправляете в базу данных, тем самым можете влиять на производительность запроса, в Entity Framework же все запросы скрыты, вы можете предавать только параметры для запроса.

Пример использования Dapper. Создаем соединение с базой данных, передаем запрос и параметры, для выполнения запроса.

```
private readonly IDbConnection db;

    public Storage(IDbConnection db)

    {

       this.db = db;
```

```
    }
public Folklorist GetFolklorist(int id)
    {
        var sql = @"SELECT * FROM [dbo].[Folklorists] WHERE Id=@Id";
        return db.Query<Folklorist>(sql, new { id }).Single();
    }
```

Dapper обеспечивает защиту от SQL инъекций, тем самым повышая безопасность сервиса.

Пример использования Dapper для документа в приложении 3.

Следующим этапом было реализовать работу REST API. Его цель предоставить клиентскому коду возможность работы с данными. Для этого в ASP.NET Core предусмотрены контроллеры. Контроллеры – это абстракция, которая позволяет в удобной форме обрабатывать входящие HTTP запросы. Контроллеры на вход принимают данные и вызывают методы для работы с данными и файлами.

В приложении контроллеры разбиты на три группы.

Первый вид контроллеров предназначен для работы с данными (DocumentApiController, FolkloristApiController, GenreApiController, InformantApiController, MTCApiController, TagApiController). Эти контроллеры передают запросы в базу данных. Основные действия: создать элемент, получить полный список элементов, обновить элемент и найти элемент.

Второй вид контроллеров предназначен для работы с бинарными файлами. Например, с электронными копиями документов. Основные действия: скачать документ и сохранить. Для хранения файлов используется Azure Blob Storage.

Третий тип позволяет делать морфологический разбор текста с помощью программы MyStem, разработанной компанией «Яндекс». На вход он принимает произвольный текст на русском языке, а возвращает список слов с лингвистической информацией, такой как начальная форма, часть речи и грамматические признаки.

Реализация контроллеров описана в приложении 2.

Для клиентской части сервиса использовался React. Главная задача React — обеспечение вывода на экран того, что можно видеть на веб-страницах. React облегчает создание интерфейсов с помощью разбиению каждой страницы на небольшие фрагменты. Они называются компонентами.

Компонент React — это участок кода, который представляет часть веб-страницы. Каждый компонент — это JavaScript функция, которая возвращает кусок кода, представляющий фрагмент страницы. Для того, чтобы сформировать страницы нужно вызывать эти функции в определённом порядке. Дальше результаты вызовов собираются вместе и показываются пользователю.

Для сервиса я использовала следующие компоненты.

DocumentCard – компонента, предназначенная для отображения карточки документа. В карточке регулируется работа над такими элементами как заголовок, содержание, год записи, информанты, фольклористы, жанры, теги, мотив или тематика, дополнительная информация, файл, место и морфологический разбор. В карточке есть несколько возможностей: создать документ, изменить документ, удалить документ и восстановить документ. В зависимости от состояния DocumentCardState и его элементов визуализируются те или иные элементы. Полный разбор карточки есть в приложении 4.

Для указания места сбора документа в карточке документа есть возможность указать точку на карте или воспользоваться поиском сервиса

«Яндекс.Карты». Для работы с сервисом используется PlaceMap компонента и библиотека react-yandex-maps.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы были проведены следующие этапы: анализ потребностей пользователей, исследование проблемы, проведение аналитической работы, сбор и обработка требований к системе, разработка плана реализации проекта, проектирование сценариев работы сервиса, проектирование базы данный, проектирование REST API, разработка ядра сервиса, проектирование интерфейса, реализация клиентской части сервиса.

Сервис удалось разместить на тестовой площадки, проведено обучение пользователей, а также собрала положительную обратную связь от заказчика проекта и первых пользователей.

Также с заказчиком запланировали следующие этапы развития сервиса: создание системы ролей и прав, настраивание логгирования ошибок сервиса, проектирование и сбор метрик сервиса, автоматизация распознавания текста, например, при загрузке электронной копии документа, создания карты для представления картины бытования фольклора во времени и пространстве и так далее.

Кроме того, CMS система в дальнейшем может быть адаптирована для создания аналогичных коллекций литературных и языковых материалов, коллекций исторических источников и так далее.

Исходя из материалов дипломной работы можно уверенно сказать, что поставленные задачи полностью решены. Цель дипломной работы достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Корпаев И. Требования для программного обеспечения: рекомендации по сбору и документированию — М.: Издательство «Книга по требованию», 2013. —118 с.
2. Зрюмов Е.А., Зрюмова А.Г. Базы данных для инженеров: Учебное пособие. — Барнаул: Изд-во АлтГТУ, 2010. – 131 с.
3. Джеймс Чамберс, Дэвид Пэкетт, Саймон Тиммс ASP.NET Core. Разработка приложений — СПб.: Питер, 2018. – 464 с.
4. Стефанов Стоян React.js. Быстрый старт — СПб.: Питер, 2017. – 304 с.

ПРИЛОЖЕНИЕ 1

```
Create Table Genres(
        Id INT NOT NULL PRIMARY KEY IDENTITY (1, 1),
        GenreName nvarchar(50));

Create Table Folklorists(
        Id INT NOT NULL PRIMARY KEY IDENTITY (1, 1),
        FIO nvarchar(200) NOT NULL);

Create Table Informants(
        Id INT NOT NULL PRIMARY KEY IDENTITY (1, 1),
        FIO nvarchar(200) NOT NULL,
        YearOfBirth INT);

Create Table MotivationalThematicClassifications(
        Id INT NOT NULL PRIMARY KEY IDENTITY (1, 1),
        Code nvarchar(10),
        ClassificationName nvarchar(100));

Create Table Tags(
        Id INT NOT NULL PRIMARY KEY IDENTITY (1, 1),
        TagName nvarchar(50));

Create Table Documents(
        Id INT NOT NULL PRIMARY KEY IDENTITY (1, 1),
        Title nvarchar(200),
        Content ntext,
        PlaceName nvarchar(200),
        PlaceLatitude float,
        PlaceLongitude float,
        YearOfRecord int,
        AdditionalInformation ntext,
```

```
        FileName nvarchar(200),
        FileId nvarchar(100),
        Deleted bit,
        CreatedAt data default GETDATE());

Create Table Morph(
        Id INT NOT NULL PRIMARY KEY IDENTITY (1, 1),
        DocumentId int,
        Analysis ntext
        FOREIGN KEY (DocumentId) REFERENCES Documents (Id)
)

Create Table Rels_Document_Informant(
        Id INT NOT NULL PRIMARY KEY IDENTITY (1, 1),
        DocumentId int,
        InformantId int
        FOREIGN KEY (DocumentId) REFERENCES Documents (Id),
        FOREIGN KEY (InformantId) REFERENCES Informants (Id));

Create Table Rels_Document_Genre(
        Id INT NOT NULL PRIMARY KEY IDENTITY (1, 1),
        DocumentId int,
        GenreId int
        FOREIGN KEY (DocumentId) REFERENCES Documents (Id),
        FOREIGN KEY (GenreId) REFERENCES Genres (Id));

Create Table Rels_Document_Folklorist(
        Id INT NOT NULL PRIMARY KEY IDENTITY (1, 1),
        DocumentId int,
        FolkloristId int
        FOREIGN KEY (DocumentId) REFERENCES Documents (Id),
        FOREIGN KEY (FolkloristId) REFERENCES Folklorists (Id));
```

```
Create Table Rels_Document_Tag(
	Id INT NOT NULL PRIMARY KEY IDENTITY (1, 1),
	DocumentId int,
	TagId int
	FOREIGN KEY (DocumentId) REFERENCES Documents (Id),
	FOREIGN KEY (TagId) REFERENCES Tags (Id));

Create Table Rels_Document_MotivationalThematicClassification(
	Id INT NOT NULL PRIMARY KEY IDENTITY (1, 1),
	DocumentId int,
	MotivationalThematicClassificationId int
	FOREIGN KEY (DocumentId) REFERENCES Documents (Id),
	FOREIGN KEY (MotivationalThematicClassificationId) REFERENCES
MotivationalThematicClassifications (Id));

GO
INSERT INTO [dbo].[Genres]
	([GenreName])
   VALUES
(N'Былина'),
...
(N'Мемората')
GO

INSERT INTO [dbo].[MotivationalThematicClassifications]
	([Code]
	,ClassificationName)
   VALUES
('A1',N' Древнее солнце'),
...
('N25',N' Путь длиной с иголку')
```

```
GO
INSERT INTO [dbo].Folklorists
        ([FIO])
    VALUES(N'Т. Нестеренко'),(N'Э. Лазарева')

GO
INSERT INTO [dbo].Informants
        ([FIO],[YearOfBirth])
    VALUES
            (N'Меньшиков Семен Андреевич',1895),(N'Шмакова Матрена Ксенофонтовна',1894)

INSERT INTO [dbo].Documents
        (Title,Content,
PlaceName,PlaceLatitude,PlaceLongitude,YearOfRecord,AdditionalInformation,FileName,FileId,Deleted)
    VALUES
            (Null,N'Ермак казак был забуяка. Он завоевывал правительству Сибирь. Он прошел по
Туре. У его дружины было первое огненное оружие, а у остальных народностей – только стрелы.
Ну, он их и брал. Про них говорили: «Если надо городишко какой взять – слободы назывались –
так он всю одежду снимет со своих казаков, сделает чучела, поставит, и сами они встанут, так
много получалось народу. Верхнюю одежду пользовали для того, чтобы запугать, чтобы сдались
татары. Царица Екатерина II полушубок подарила Ермаку, когда он Сибирь брал. В Верхотурьи
крепость осталась. Ее построили или татары, или русские от набегов татар уже после
Ермака.',N'Деревня Кузино Алапаевского района',57.849877,62.157731,1966,null,null,null,0)

INSERT INTO [dbo].[Rels_Document_Folklorist]
            ([DocumentId],[FolkloristId])
            values   (1,1),(1,2)

INSERT INTO [dbo].[Rels_Document_Informant]
            ([DocumentId],[InformantId])
            values   (1,1)

INSERT INTO [dbo].[Rels_Document_Genre]
            ([DocumentId],[GenreId])
            values   (1,24)
```

```
Select
d.Id,d.Title,d.Content,d.PlaceName,d.YearOfRecord,i.FIO,f.FIO,d.AdditionalInformation,d.FileName,d.Del
eted

From [dbo].[Documents] d

inner join [dbo].[Rels_Document_Folklorist] rdf on d.Id=rdf.DocumentId

inner join [dbo].[Rels_Document_Informant] rdi on d.Id=rdi.DocumentId

inner join [dbo].[Folklorists] f on rdf.FolkloristId=f.Id

inner join [dbo].[Informants] i on rdi.InformantId=i.Id


UPDATE [dbo].[Documents]   SET Deleted = 0  WHERE Id=1


DELETE FROM [dbo].[Documents]      WHERE Id = 2

GO


INSERT INTO [dbo].[Morph] ([DocumentId],[Analysis])     VALUES (24,'')
```

ПРИЛОЖЕНИЕ 2

```
namespace Folklore.Controllers
{
    [Route("api/document")]
    public class DocumentApiController : ControllerBase
    {
        private readonly IStorage storage;

        public DocumentApiController(IStorage storage)
        {
            this.storage = storage;
        }

        public Document GetDocument([FromQuery] int id)
        {
            return storage.GetDocument(id);
        }
        [Route("all")]
        [HttpGet]
        public IEnumerable<Document> GetAllDocuments()
        {
            return storage.GetAllDocuments();
        }

        [HttpPost]
        public Document AddDocument([FromBody]Document newDocument)
        {
            return storage.AddDocument(newDocument);
        }

        [HttpDelete]
        public void DeleteDocument([FromQuery] int id)
        {
            storage.DeleteDocument(id);
        }

        [HttpPatch]
        public Document UpdateDocument([FromBody] Document updateDocument)
        {
            return storage.UpdateDocument(updateDocument);
        }
        [Route("search")]
        [HttpGet]
        public IEnumerable<Document> SearchDocument([FromQuery] string param, string
genre, string place, string yearOfRecord, string informant)
        {
            return storage.SearchDocument(param,genre,place, yearOfRecord,informant);
        }
    }
}

namespace Folklore.Controllers
{
    [Route("api/folklorist")]
    public class FolkloristApiController : ControllerBase
    {
        private readonly IStorage storage;

        public FolkloristApiController(IStorage storage)
        {
            this.storage = storage;
        }

        [HttpGet]
```

```
        public IEnumerable<Folklorist> GetAllFolklorists()
        {
            return storage.GetAllFolklorists();
        }
        [Route("search")]
        [HttpGet]
        public IEnumerable<Folklorist> SearchFolklorist([FromQuery] string folklorist)
        {
            return storage.SearchFolklorist(folklorist);
        }

        [HttpPost]
        public Folklorist AddFolklorist([FromBody] Folklorist newFolklorist)
        {
            return storage.AddFolklorist(newFolklorist);
        }
    }
}

namespace Folklore.Controllers
{
    [Route("api/genre")]
    public class GenreApiController : ControllerBase
    {
        private readonly IStorage storage;

        public GenreApiController(IStorage storage)
        {
            this.storage = storage;
        }

        [HttpGet]
        public IEnumerable<Genres> GetAllGenres()
        {
            return storage.GetAllGenres();
        }
        [Route("search")]
        [HttpGet]
        public IEnumerable<Genres> SearchGenres([FromQuery] string genre)
        {
            return storage.SearchGenres(genre);
        }

        [HttpPost]
        public Genres AddGenre([FromBody] Genres newGenres)
        {
            return storage.AddGenre(newGenres);
        }
    }
}

namespace Folklore.Controllers
{
    [Route("api/informant")]
    public class InformantApiController : ControllerBase
    {
        private readonly IStorage storage;

        public InformantApiController(IStorage storage)
        {
            this.storage = storage;
        }

        [HttpGet]
        public IEnumerable<Informant> GetAllFolklorists()
```

```
        {
            return storage.GetAllInformants();
        }

        [Route("search")]
        [HttpGet]
        public IEnumerable<Informant> SearchInformants([FromQuery] string informant)
        {
            return storage.SearchInformants(informant);
        }

        [HttpPost]
        public Informant AddInformant([FromBody] Informant newInformant)
        {
            return storage.AddInformant(newInformant);
        }
    }
}

namespace Folklore.Controllers
{
    [Route("api/morph")]
    public class MorphApiController : ControllerBase
    {
        public readonly IMystemClient mystem;

        public MorphApiController(IMystemClient mystem)
        {
            this.mystem = mystem;
        }


        [HttpPost]
        public List<MorphInfo> Upload([FromBody]MorphRequest morphRequest)
        {
            var morphs = mystem.Run(morphRequest.Text ?? "").ToList();

            return morphs;
        }
    }
}

namespace Folklore.Controllers
{
    [Route("api/mtc")]
    public class MTCApiController : ControllerBase
    {
        private readonly IStorage storage;

        public MTCApiController(IStorage storage)
        {
            this.storage = storage;
        }

        [HttpGet]
        public IEnumerable<MotivationalThematicClassification> GetAllMTCs()
        {
            return storage.GetAllMTCs();
        }
        [Route("search")]
        [HttpGet]
        public IEnumerable<MotivationalThematicClassification> SearchMTCs([FromQuery]
string mtc, string code)
        {
            return storage.SearchMTCs(mtc, code);
```

```
        }

        [HttpPost]
        public MotivationalThematicClassification AddMTC([FromBody]
MotivationalThematicClassification newMTC)
        {
            return storage.AddMTC(newMTC);
        }
    }
}

namespace Folklore.Controllers
{
    [Route("api/tag")]
    public class TagApiController : ControllerBase
    {
        private readonly IStorage storage;

        public TagApiController(IStorage storage)
        {
            this.storage = storage;
        }

        [HttpGet]
        public IEnumerable<Tag> GetAllTags()
        {
            return storage.GetAllTags();
        }

        [HttpPost]
        public Tag AddTag([FromBody] Tag newTag)
        {
            return storage.AddTag(newTag);
        }
        [Route("search")]
        [HttpGet]
        public IEnumerable<Tag> SearchTags([FromQuery] string tag)
        {
            return storage.SearchTags(tag);
        }
    }
}

namespace Folklore.Controllers
{
    [Route("api/file")]
    public class FileApiController : ControllerBase
    {
        private readonly IFileStorage fileStorage;
        private static readonly IContentTypeProvider contentTypeProvider = new
FileExtensionContentTypeProvider();

        public FileApiController(IFileStorage fileStorage)
        {
            this.fileStorage = fileStorage;
        }

        [HttpGet]
        public FileResult Download(string id)
        {
            var (data, name) = fileStorage.Download(id);

            return new FileStreamResult(data, GetContentType(name))
            {
                FileDownloadName = name
```

```
            };
        }

        [HttpPost]
        public object Upload([FromQuery]string name)
        {
            using var body = Request.Body;
            var id = fileStorage.Upload(body, name);

            return new {id};
        }

        private static string GetContentType(string fileName)
        {
            return !contentTypeProvider.TryGetContentType(fileName, out var contentType)
                ? "application/octet-stream"
                : contentType;
        }
    }
}
```

ПРИЛОЖЕНИЕ 3

```
namespace Folklore.Storages
{
    public class Storage : IStorage
    {
        private readonly IDbConnection db;

        public Storage(IDbConnection db)
        {
            this.db = db;
        }

        #region Document

        public IEnumerable<Document> GetAllDocuments()
        {
            return db.Query<Document>(@"Select TOP(200) * From [dbo].[Documents] WHERE
Deleted = 0 order by [CreatedAt] desc");
        }

        public Document GetDocument(int id)
        {
            var doc = db.Query<Document>(@"
Select *
From [dbo].[Documents] d
Where d.Id=@Id
", new {Id = id}).Single();

            var sqlrdf = @"Select f.*
From [dbo].[Documents] d
inner join [dbo].[Rels_Document_Folklorist] rdf on d.Id=rdf.DocumentId
inner join [dbo].[Folklorists] f on rdf.FolkloristId=f.Id
Where d.Id=@Id";
            var folklorists = db.Query<Folklorist>(sqlrdf, new {Id=id});
            doc.Folklorists.AddRange(folklorists);

            var sqlrdi = @"Select i.*
From [dbo].[Documents] d
inner join [dbo].[Rels_Document_Informant] rdi on d.Id=rdi.DocumentId
inner join [dbo].[Informants] i on rdi.InformantId=i.Id
Where d.Id=@Id";
            var informants = db.Query<Informant>(sqlrdi, new {Id = id});
            doc.Informants.AddRange(informants);

            var sqlrdg = @"Select g.*
From [dbo].[Documents] d
inner join [dbo].[Rels_Document_Genre] rdg on rdg.DocumentId=d.Id
inner join [dbo].[Genres] g on rdg.[GenreId]=g.Id
Where d.Id=@Id";
            var genres = db.Query<Genres>(sqlrdg, new { Id = id });
            doc.Genres.AddRange(genres);


            var sqlrdmtc = @"Select mtc.*
From [dbo].[Documents] d
inner join [dbo].[Rels_Document_MotivationalThematicClassification] rdmtc on
rdmtc.DocumentId=d.Id
inner join [dbo].[MotivationalThematicClassifications] mtc on
mtc.[Id]=rdmtc.[MotivationalThematicClassificationId]
Where d.Id=@Id";
            var mtcs = db.Query<MotivationalThematicClassification>(sqlrdmtc, new { Id =
id });
            doc.MotivationalThematicClassifications.AddRange(mtcs);
```

```
            var sqlrdt = @"Select t.*
From [dbo].[Documents] d
inner join [dbo].[Rels_Document_Tag] rdt on rdt.[DocumentId]=d.Id
inner join [dbo].[Tags] t on rdt.[TagId]=t.Id
Where d.Id=@Id";
            var tags = db.Query<Tag>(sqlrdt, new { Id = id });
            doc.Tags.AddRange(tags);

            var m = db.Query<string>(@"
Select Analysis From [dbo].[Morph] d
Where d.DocumentId=@Id
", new { Id = id }).FirstOrDefault();

            doc.Morph = m;
            return doc;
        }

        public Document UpdateDocument(Document updateDocument)
        {
            var sqlUpdDoc = @"
UPDATE [dbo].[Documents]
SET Title=@Title,
    Content=@Content,
    PlaceName=@PlaceName,
    PlaceLatitude=@PlaceLatitude,
    PlaceLongitude=@PlaceLongitude,
    YearOfRecord=@YearOfRecord,
    AdditionalInformation=@AdditionalInformation,
    FileName=@FileName,
    FileId=@FileId
WHERE Id=@Id";
            db.Execute(sqlUpdDoc, new
            {
                Title = updateDocument.Title,
                Content = updateDocument.Content,
                PlaceName = updateDocument.PlaceName,
                PlaceLatitude = updateDocument.PlaceLatitude,
                PlaceLongitude = updateDocument.PlaceLongitude,
                YearOfRecord = updateDocument.YearOfRecord,
                AdditionalInformation = updateDocument.AdditionalInformation,
                FileName = updateDocument.FileName,
                FileId=updateDocument.FileId,
                Id = updateDocument.Id
            });

            UpdateInformants(updateDocument);
            UpdateFolklorists(updateDocument);
            UpdateTags(updateDocument);
            UpdateGenres(updateDocument);
            UpdateMTC(updateDocument);
            UpdateMorph(updateDocument);

            return GetDocument(updateDocument.Id);
        }
```

ПРИЛОЖЕНИЕ 4

```
import * as React from 'react';
import DocumentApi from '../api/documentsApi';
import FolkDocument from '../models/Document';
import { Col, Button, Input, Badge, UncontrolledTooltip } from 'reactstrap';
import { Informant } from '../models/Informant';
import { InputRow } from '../components/InputRow';
import { DocInput } from '../components/DocInput';
import { Tag } from '../models/Tag';
import { Folklorist } from '../models/Folklorist';
import { Genre } from '../models/Genre';
import { MotivationalThematicClassification } from '../models/MotivationalThematicClassification';
import PlaceMap, { PlaceInfo } from './PlaceMap';
import { AsyncTypeahead } from 'react-bootstrap-typeahead';
import { parseMorphCsv, serializeMorphCsv } from '../utils/morph';
import { MorphInfo } from "../models/MorphInfo";

import './card.css';
import 'react-bootstrap-typeahead/css/Typeahead.css';
import { TableEditor } from './TableEditor';
import { downloadTextFile, uploadTextFile } from '../utils/files';

export interface DocumentCardProps {
  doc: FolkDocument;
  editing: boolean;
  onDocSave: (doc: FolkDocument) => Promise<void>;
}

export interface DocumentCardState {
  doc: FolkDocument;
  editing: boolean;
  newInformant: Informant;
```

```
  newFolklorist: Folklorist;
  newTag: Tag;
  newGenre: Genre;
  newMTC: MotivationalThematicClassification;
  valid: boolean;
  morph: MorphInfo[];

  // for search
  informants: Informant[];
  searchLoadingInformant: boolean;
  searchLoadingFolklorist: boolean;
  searchLoadingTag: boolean;
  searchLoadingGenre: boolean;
  searchLoadingMTC: boolean;
  folklorists: Folklorist[];
  genres: Genre[];
  mtcs: MotivationalThematicClassification[];
  tags: Tag[];
}

export default class DocumentCard extends React.Component<DocumentCardProps, DocumentCardStat
e> {
  constructor(props: DocumentCardProps, state: DocumentCardState) {
    super(props, state);
    this.state = {
      doc: props.doc,
      editing: props.editing,
      newInformant: {
        fio: ''
      },
      newFolklorist: {
        fio: ''
```

```
    },
    newTag: {
      tagName: ''
    },
    newGenre: {
      genreName: ''
    },
    newMTC: {
      code: '',
      classificationName: ''
    },
    valid: false,
    informants: [],
    morph: parseMorphCsv(props.doc.morph),

    // search
    searchLoadingInformant: false,
    searchLoadingFolklorist: false,
    searchLoadingTag: false,
    searchLoadingGenre: false,
    searchLoadingMTC: false,
    folklorists: [],
    genres: [],
    mtcs: [],
    tags: [],
  };
}

startEditing() {
  this.setState({ editing: true });
}
```

```
async saveChanges() {
  let { doc } = this.state;
  doc = {
    ...doc,
    morph: serializeMorphCsv(this.state.morph) // сохраянем редактированный morph
  }
  this.setState({ editing: false });
  await this.props.onDocSave(doc);
}

checkTitle() {
  let { doc } = this.state;
  let inputs: string[] = []

  if (doc.title === '' || doc.title.length > 200) {
    inputs.push('title')
  }
  if (doc.yearOfRecord < 1000 || doc.yearOfRecord > 9999) {
    inputs.push('yearOfRecord')
  }
  return inputs

}

checkUniqeId(array: {id?:number}[], id:number|undefined):boolean{
  if (!id){
    return true;
  }

  return array.every(x=>x.id !== id);
}
```

```
 checkMinForms() {

   let { newFolklorist, newInformant, newTag, newGenre, newMTC, doc } = this.state;

   let inputs: string[] = []


   if (newFolklorist.fio === '' || newFolklorist.fio.length > 200 || !this.checkUniqeId(doc.folklorists,newF
olklorist.id)) {

     inputs.push('newFolklorist')

   }

   if (newInformant.fio === '' || newInformant.fio.length > 200 || !this.checkUniqeId(doc.informants,ne
wInformant.id)) {

     inputs.push('newInformant')

   }

   if (newInformant.yearOfBirth && (newInformant.yearOfBirth < 1000 || newInformant.yearOfBirth > 9
999)) {

     inputs.push('newInformant')

   }

   if (newTag.tagName === '' || newTag.tagName.length > 50 || !this.checkUniqeId(doc.tags,newTag.id))
 {

     inputs.push('newTag')

   }

   if (newGenre.genreName === '' || newGenre.genreName.length > 50 || !this.checkUniqeId(doc.genre
s,newGenre.id)) {

     inputs.push('newGenre')

   }

   if (newMTC.classificationName === '' || newMTC.classificationName.length > 50 || !this.checkUniqeId
(doc.motivationalThematicClassifications,newMTC.id)) {

     inputs.push('newMTC')

   }

   return inputs


 }


 changeDocument(partialDocument: Partial<FolkDocument>) {

   let { doc } = this.state;
```

```
  this.setState({
    doc: {
      ...doc,
      ...partialDocument
    }
  });
}

changeNewInformant(partialInformant: Partial<Informant>) {
  let { newInformant } = this.state;

  this.setState({
    newInformant: {
      ...newInformant,
      ...partialInformant
    }
  });
}

pushNewInformant() {
  let { newInformant, doc } = this.state;

  this.changeDocument({
    informants: [
      ...doc.informants,
      newInformant
    ]
  });
  this.setState({
    newInformant: {
      fio: ''
```

```
    }
  });
}

deleteInformant(deleteIndex: number) {
  let { doc } = this.state;

  this.changeDocument({
    informants: doc.informants.filter((_, i) => i !== deleteIndex)
  });
}

async searchInformants(q: string) {
  this.setState({ searchLoadingInformant: true })
  this.setState({ informants: await DocumentApi.searchInformants(q) })
  this.setState({ searchLoadingInformant: false })
}

renderInformant() {
  const { doc, editing, newInformant, informants, searchLoadingInformant } = this.state;

  const informantComponents = doc.informants.map((informant, i) => (
    <InputRow key={i}>
      <Col sm={8}>
        <Input readOnly={true} type="text" value={informant.fio || ''} />
      </Col>
      <Col>
        <Input readOnly={true} type="number" value={informant.yearOfBirth || ''} />
      </Col>
      {editing
        ? <Col>
```

```
        <Button outline onClick={() => this.deleteInformant(i)} color="danger" style={{ width: "100%" }}>У
далить</Button>
        </Col>
        : null}
    </InputRow>
  ));

  let recommendation = this.checkMinForms();
  let ref = React.createRef<any>()
  const editingComponent = (
    <InputRow>
      <Col sm={8}>
        <AsyncTypeahead
          id="labelkey-example"
          labelKey="fio"
          onSearch={(q) => this.searchInformants(q)}
          isLoading={searchLoadingInformant}
          minLength={3}
          options={informants}
          placeholder="ФИО"
          onInputChange={e => this.changeNewInformant({ fio: e, id: undefined })}
          onChange={e => e.length > 0 ? this.changeNewInformant({ id: e[0].id, fio: e[0].fio, yearOfBirth: e[
0].yearOfBirth }) : ''}
          renderMenuItemChildren={(option) => (
            <div>
              <span>{option.fio} {option.yearOfBirth && `${option.yearOfBirth}г.р.`}</span>
            </div>
          )}
          ref={ref}
        /></Col>

      <Col sm={2}>
        <Input
```

```
          type="number"
          placeholder="Год"
          value={newInformant.yearOfBirth || ''}
          onChange={e => this.changeNewInformant({ yearOfBirth: parseInt(e.target.value, 10) || undefine
d, id: undefined})}
        />
      </Col>
      <Col sm={2}>
        <Button outline color="primary" style={{ width: "100%" }} disabled={recommendation.includes('ne
wInformant')} onClick={() => { this.pushNewInformant(); ref.current.clear(); }}>Добавить</Button>
      </Col>
    </InputRow>
  );

  return (
    <React.Fragment>
      {informantComponents}
      {editing ? editingComponent : null}
    </React.Fragment>
  );
 }

 changeNewFolklorist(partialFolklorist: Partial<Folklorist>) {
  let { newFolklorist } = this.state;

  this.setState({
    newFolklorist: {
      ...newFolklorist,
      ...partialFolklorist
    }
  });
 }
```

```
pushNewFolklorist() {
  let { newFolklorist, doc } = this.state;

  this.changeDocument({
    folklorists: [
      ...doc.folklorists,
      newFolklorist
    ]
  });
  this.setState({
    newFolklorist: {
      fio: ''
    }
  });
}

deleteFolklorist(deleteIndex: number) {
  let { doc } = this.state;

  this.changeDocument({
    folklorists: doc.folklorists.filter((_, i) => i !== deleteIndex)
  });
}

async searchFolklorists(q: string) {
  this.setState({ searchLoadingFolklorist: true })
  this.setState({ folklorists: await DocumentApi.searchFolklorist(q) })
  this.setState({ searchLoadingFolklorist: false })
}

renderFolklorist() {
  const { doc, editing, searchLoadingFolklorist, folklorists } = this.state;
```

```
  const folkloristComponents = doc.folklorists.map((folklorists, i) => (
    <InputRow key={i}>
      <Col sm={editing ? 10 : 12}>
        <Input readOnly={true} type="text" value={folklorists.fio} />
      </Col>
      {editing
        ? <Col>
          <Button outline onClick={() => this.deleteFolklorist(i)} color="danger" style={{ width: "100%" }}>У
далить</Button>
        </Col>
        : null}
    </InputRow>
  ));

  let recommendation = this.checkMinForms();
  let ref = React.createRef<any>()
  const editingComponent = (
    <InputRow>
      <Col sm={10}>
        <AsyncTypeahead
          id="labelkey-example"
          labelKey="fio"
          onSearch={(q) => this.searchFolklorists(q)}
          isLoading={searchLoadingFolklorist}
          minLength={3}
          options={folklorists}
          placeholder="ФИО"
          onInputChange={e => this.changeNewFolklorist({ fio: e ,id: undefined})}
          onChange={e => e.length > 0 ? this.changeNewFolklorist({ id: e[0].id, fio: e[0].fio }) : ''}
          renderMenuItemChildren={(option) => (
            <div>
```

```
            <span> {option.fio}</span>
          </div>
         )}
         ref={ref}
        /></Col>
      <Col sm={2}>
       <Button outline color="primary" style={{ width: "100%" }} disabled={recommendation.includes('ne
wFolklorist')} onClick={() => { this.pushNewFolklorist(); ref.current.clear(); }}>Добавить</Button>
      </Col>
    </InputRow>
  );

  return (
    <React.Fragment>
      {folkloristComponents}
      {editing ? editingComponent : null}
    </React.Fragment>
  );
}

 changeNewTag(partialTag: Partial<Tag>) {
  let { newTag } = this.state;

  this.setState({
    newTag: {
      ...newTag,
      ...partialTag
    }
  })
 }

 pushNewTag() {
```

```
  let { newTag, doc } = this.state;

  this.changeDocument({
    tags: [
      ...doc.tags,
      newTag
    ]
  });
  this.setState({
    newTag: {
      tagName: ''
    }
  });
}
deleteTag(deleteIndex: number) {
  let { doc } = this.state;
  this.changeDocument({
    tags: doc.tags.filter((_, i) => i !== deleteIndex)
  })
}
makeBadge(tagName: string, i: number) {
  const { editing } = this.state;
  if (editing) {
    return (
      <React.Fragment key={i}>
        <Badge id={`tag${i}`} color="primary" style={{ margin: "3px", fontSize: "12pt", cursor: "pointer" }} o
nClick={() => { this.deleteTag(i) }}>
          {tagName}
        </Badge>
        <UncontrolledTooltip placement="bottom" target={`tag${i}`}>
          Нажмите, чтобы удалить
    </UncontrolledTooltip>
```

```
    </React.Fragment>);
  }

  return (<Badge key={i} color="primary" style={{ margin: "3px", fontSize: "12pt" }}>
    {tagName}
  </Badge>);
}

async searchTags(q: string) {
  this.setState({ searchLoadingTag: true })
  this.setState({ tags: await DocumentApi.searchTags(q) })
  this.setState({ searchLoadingTag: false })
}

renderTags() {
  const { doc, editing, tags, searchLoadingTag } = this.state;
  let t = doc.tags.map((tag, i) =>
    this.makeBadge(tag.tagName, i)
  );
  let recommendation = this.checkMinForms();
  let ref = React.createRef<any>()
  let editingComponent = (

    <InputRow>

      <Col sm={10}>
        <AsyncTypeahead
          id="labelkey-example"
          labelKey="tagName"
          onSearch={(q) => this.searchTags(q)}
          isLoading={searchLoadingTag}
          minLength={3}
```

```
          options={tags}
          placeholder="Ter"
          onInputChange={e => this.changeNewTag({ tagName: e, id: undefined})}
          onChange={e => e.length > 0 ? this.changeNewTag({ id: e[0].id, tagName: e[0].tagName }) : ''}
          renderMenuItemChildren={(option) => (
            <div>
              <span> {option.tagName}</span>
            </div>
          )}
          ref={ref}
        /></Col>

      <Col sm={2}>
        <Button outline color="primary" style={{ width: "100%" }} disabled={recommendation.includes('ne
wTag')} onClick={() => { this.pushNewTag(); ref.current.clear(); }}>Добавить</Button>
      </Col>
    </InputRow>
  );

  return (
    <React.Fragment>
      <InputRow>
        <Col>{t}</Col>
      </InputRow>
      {editing ? editingComponent : null}
    </React.Fragment>
  );
 }

 //genre
 changeNewGenre(partialGenre: Partial<Genre>) {
  let { newGenre } = this.state;
```

```
  this.setState({
    newGenre: {
      ...newGenre,
      ...partialGenre
    }
  })
}

pushNewGenre() {
  let { newGenre, doc } = this.state;

  this.changeDocument({
    genres: [
      ...doc.genres,
      newGenre
    ]
  });
  this.setState({
    newGenre: {
      genreName: ''
    }
  });
}
deleteGenre(deleteIndex: number) {
  let { doc } = this.state;
  this.changeDocument({
    genres: doc.genres.filter((_, i) => i !== deleteIndex)
  })
}
makeBadgeGenre(genreName: string, i: number) {
  const { editing } = this.state;
```

```
  if (editing) {
    return (
      <React.Fragment key={i}>
        <Badge id={`genre${i}`} color="primary" style={{ margin: "3px", fontSize: "12pt", cursor: "pointer" }
} onClick={() => { this.deleteGenre(i) }}>
          {genreName}
        </Badge>
        <UncontrolledTooltip placement="bottom" target={`genre${i}`}>
          Нажмите, чтобы удалить
        </UncontrolledTooltip>
      </React.Fragment>);
  }

  return (<Badge key={i} color="primary" style={{ margin: "3px", fontSize: "12pt" }}>
    {genreName}
  </Badge>);
}

async searchGenres(q: string) {
  this.setState({ searchLoadingGenre: true })
  this.setState({ genres: await DocumentApi.searchGenres(q) })
  this.setState({ searchLoadingGenre: false })
}

renderGenres() {
  const { doc, editing, searchLoadingGenre, genres } = this.state;
  let g = doc.genres.map((genre, i) =>
    this.makeBadgeGenre(genre.genreName, i)
  );
  let recommendation = this.checkMinForms();
  let ref = React.createRef<any>()
  let editingComponent = (
```

```
<InputRow>
 <Col sm={10}>
  <AsyncTypeahead
   id="labelkey-example"
   onSearch={(q) => this.searchGenres(q)}
   labelKey = 'genreName'
   isLoading={searchLoadingGenre}
   minLength={3}
   options={genres}
   placeholder="Жанр"
   onInputChange={e => this.changeNewGenre({ genreName: e,id: undefined })}
   onChange={e => {
    if (e.length > 0) {
     this.changeNewGenre({ genreName: e[0].genreName, id: e[0].id})
    }
   }}
   renderMenuItemChildren={(option) => (
    <div>
     <span> {option.genreName}</span>
    </div>
   )}
   ref={ref}
  /></Col>
 <Col sm={2}>
  <Button
   outline
   color="primary"
   style={{ width: "100%" }}
   disabled={recommendation.includes('newGenre')}
   onClick={() => {
    this.pushNewGenre();
    ref.current.clear();
```

```
        }}
       >
        Добавить
     </Button>
     </Col>
   </InputRow>
  );

  return (
   <React.Fragment>
     <InputRow>
       <Col>{g}</Col>
     </InputRow>
     {editing ? editingComponent : null}
   </React.Fragment>
  );
}

//мтк
changeNewMTC(partialMTC: Partial<MotivationalThematicClassification>) {
  let { newMTC } = this.state;

  this.setState({
    newMTC: {
     ...newMTC,
     ...partialMTC
    }
  });
}

pushNewMTC() {
  let { newMTC, doc } = this.state;
```

```
  this.changeDocument({
    motivationalThematicClassifications: [
      ...doc.motivationalThematicClassifications,
      newMTC
    ]
  });
  this.setState({
    newMTC: {
      code: '',
      classificationName: ''
    }
  });
 }

 deleteMTC(deleteIndex: number) {
  let { doc } = this.state;

  this.changeDocument({
    motivationalThematicClassifications: doc.motivationalThematicClassifications.filter((_, i) => i !== dele
teIndex)
  });
 }

 async searchMTCs(q: string) {
  this.setState({ searchLoadingMTC: true })
  this.setState({ mtcs: await DocumentApi.searchMTCs(q) })
  this.setState({ searchLoadingMTC: false })
 }

 renderMTC() {
  const { doc, editing, newMTC, searchLoadingMTC, mtcs } = this.state;
```

```
  const mtcComponents = doc.motivationalThematicClassifications.map((mtc, i) => (
    <InputRow key={i}>
      <Col sm={8}>
        <Input readOnly={true} type="text" value={mtc.classificationName} />
      </Col>
      <Col>
        <Input readOnly={true} type="text" value={mtc.code} />
      </Col>
      {editing
        ? <Col>
          <Button outline onClick={() => this.deleteMTC(i)} color="danger" style={{ width: "100%" }}>Удали
ть</Button>
        </Col>
        : null}
    </InputRow>
  ));

  let recommendation = this.checkMinForms();
  let ref = React.createRef<any>()
  const editingComponent = (
    <InputRow>
      <Col sm={8}>
        <AsyncTypeahead
          id="labelkey-example"
          labelKey="classificationName"
          onSearch={(q) => this.searchMTCs(q)}
          isLoading={searchLoadingMTC}
          minLength={3}
          options={mtcs}
          placeholder="Мотив или тематика"
          onInputChange={e => this.changeNewMTC({ classificationName: e,id: undefined })}
```

```
        onChange={e => e.length > 0 ? this.changeNewMTC({ id: e[0].id, classificationName: e[0].classificationName, code: e[0].code }) : ''}
        renderMenuItemChildren={(option) => (
          <div>
            <span>{option.classificationName} ({option.code})</span>
          </div>
        )}
        ref={ref}
       /></Col>
      <Col sm={2}>
       <Input type="text" placeholder="Код" value={newMTC.code} onChange={e => this.changeNewMTC({ code: e.target.value, id: undefined})} />
      </Col>
      <Col sm={2}>
       <Button outline color="primary" style={{ width: "100%" }} disabled={recommendation.includes('newMTC')} onClick={() => { this.pushNewMTC(); ref.current.clear(); }}>Добавить</Button>
      </Col>
    </InputRow>
  );

  return (
    <React.Fragment>
      {mtcComponents}
      {editing ? editingComponent : null}
    </React.Fragment>
  );
 }

 renderPlace() {
  const { editing, doc } = this.state;

  let coords = doc.placeLatitude && doc.placeLongitude ? [doc.placeLatitude, doc.placeLongitude] : undefined;
```

```
  let onPlaceInfo = (placeInfo: PlaceInfo) => {
    this.changeDocument({
      placeName: placeInfo.addressLine,
      placeLatitude: placeInfo.latitude,
      placeLongitude: placeInfo.longtitude
    });
  }

  let editingComponent = (
    <InputRow>
      <Col>
        <PlaceMap onPlaceInfo={onPlaceInfo} coords={coords} />
      </Col>
    </InputRow>
  );

  return (
    <React.Fragment>
      <InputRow>
        <Col>
          <Input readOnly={true} type="text" value={doc.placeName || ''} onChange={x => this.changeDocu
ment({ placeName: x.target.value })} />
        </Col>
      </InputRow>
      {editing ? editingComponent : null}
    </React.Fragment>

  );
 }

 initializeUploadDocumentFile() {
  const file = document.getElementById("uploadDocument");
```

```
  if (file) {
    file.click();
  }
}

async uploadDocumentFile(fileList: FileList | null) {
  if (!fileList || fileList.length === 0) {
    return;
  }

  const file = fileList[0];
  const id = await DocumentApi.uploadFile(file);
  this.changeDocument({ fileName: file.name, fileId: id });
}

renderContent() {
  const { doc: { content, morph }, editing } = this.state;
  if (editing) {
    return <Input
      style={{ height: "200px", resize: "none" }}
      type="textarea"
      value={content || ''}
      onChange={x => this.changeDocument({ content: x.target.value })}
    />;
  }

  if (!content) {
    return;
  }

  let wordRegex = /[а-яА-ЯёЁ-]+/g;
  let delimiter = '$$';
```

```
let contentWithDelimiters = content.replace(wordRegex, str => `${delimiter}${str}${delimiter}`);

const morphs = parseMorphCsv(morph);
const parts = contentWithDelimiters.split(delimiter);

const renderedParts: any[] = [];
let currentWord = 0;
parts.forEach((part, i) => {
 if (!wordRegex.test(part)) {
  renderedParts.push(part);
  return;
 }

 const morph: MorphInfo = morphs.length > currentWord
  ? morphs[currentWord]
  : { word: '', initialForm: '', partOfSpeach: '', grammaticalSigns: '' };

 const id = `word${currentWord}`;
 const renderedPart = (
  <React.Fragment key={i}>
   <span className="word" id={id}>{part}</span>
   <UncontrolledTooltip style={{ textAlign: "left" }} placement="bottom" target={id}>
    <span style={{ fontStyle: "italic" }}>Начальная форма:</span> {morph.initialForm}<br />
    <span style={{ fontStyle: "italic" }}>Часть речи:</span> {morph.partOfSpeach} <br />
    <span style={{ fontStyle: "italic" }}>Признаки:</span> {morph.grammaticalSigns} <br />
   </UncontrolledTooltip>
  </React.Fragment>
 );
 renderedParts.push(renderedPart);
 currentWord++;
})
```

```
  return <p style={{ whiteSpace: "pre-line" }}>{renderedParts}</p>
}

exportMorphCsv() {
  const csv = serializeMorphCsv(this.state.morph);
  const name = this.state.doc.title.replace(/\s+/g, '_');
  downloadTextFile(`${name}.morph.csv`, csv);
}

importMorphCsv() {
  uploadTextFile(csv => {
    const morph = parseMorphCsv(csv);
    this.setState({ morph });
  }, '.csv');
}

async getMorphsForText() {
  const { doc } = this.state;
  const text = doc.content || "";
  if (/^\s+$/g.test(text)) {
    return;
  }
  const morph = await DocumentApi.getMorphs(text);
  this.setState({ morph })
}

renderMorph() {
  const { doc } = this.state;

  return (
    <>
      <InputRow>
```

```
        <Col>
          <Button outline style={{width: '100%'}} onClick={() => this.exportMorphCsv()}>Экспорт в CSV</Button>
        </Col>
        <Col>
          <Button outline style={{width: '100%'}} onClick={() => this.importMorphCsv()}>Импорт из CSV</Button>
        </Col>
        <Col>
          <Button outline style={{width: '100%'}} onClick={() => this.getMorphsForText()}>Заполнить</Button>
        </Col>
      </InputRow>
      <InputRow>

        <Col>
          <TableEditor
            header={['Слово', 'Начальная форма', 'Часть речи', 'Граматические признаки']}
            sizes={[20, 20, 15, 45]}
            columns={['word', 'initialForm', 'partOfSpeach', 'grammaticalSigns']}
            data={this.state.morph}
            onChange={(row, col, value) => {
              const newMorph = [...this.state.morph]
              newMorph[row] = {
                ...newMorph[row],
                [col]: value.trim()
              }
              this.setState({ morph: newMorph });
            }}
            renderCell={(item, key) => item[key]}
          />
        </Col>
      </InputRow>
```

```
    </>
  );
 }

 render() {
  const { doc, editing } = this.state;
  let block = this.checkTitle();

  return (
   <div>
    <DocInput label="Название">
     <Input readOnly={!editing} value={doc.title} invalid={block.includes('title')} onChange={x => this.ch
angeDocument({ title: x.target.value })} />
    </DocInput>

    <DocInput
     visible={doc.yearOfRecord !== 0 || editing}
     label="Год записи">
     <Input readOnly={!editing} type="number" value={doc.yearOfRecord} invalid={block.includes('year
OfRecord')} onChange={x => this.changeDocument({ yearOfRecord: parseInt(x.target.value, 10) || 2020 }
)} />
    </DocInput>

    <DocInput
     visible={doc.content !== '' || editing}
     label="Содержание">
      <div>
       {this.renderContent()}
      </div>
    </DocInput>

    <DocInput
```

```
  visible={doc.informants.length > 0 || editing}
  label="Информанты">
  {this.renderInformant()}
</DocInput>

<DocInput
  visible={doc.folklorists.length > 0 || editing}
  label="Фольклористы">
  {this.renderFolklorist()}
</DocInput>

<DocInput
  visible={doc.tags.length > 0 || editing}
  label="Теги">
  {this.renderTags()}
</DocInput>

<DocInput
  visible={doc.genres.length > 0 || editing}
  label="Жанры">
  {this.renderGenres()}
</ DocInput>

<DocInput
  visible={doc.motivationalThematicClassifications.length > 0 || editing}
  label="Мотивно-тематический классификатор">
  {this.renderMTC()}
</ DocInput>

<DocInput
  visible={doc.additionalInformation !== '' || editing}
  label="Дополнительная информация">
```

```
      <Input readOnly={!editing} style={{ height: "100px", resize: "none" }} type="textarea" value={doc.a
dditionalInformation || ''} onChange={x => this.changeDocument({ additionalInformation: x.target.value
 })} />
    </DocInput>

    <DocInput
     visible={doc.fileName !== '' || editing}
     label="Документ">
     <InputRow>
      <Col sm={10}>
       <a href={DocumentApi.makeDocumentFileDownloadLink(doc.fileId || "")}>{doc.fileName}</a>
      </Col>
      {
       editing
        ? <Col sm={2}>
          <input type="file" id="uploadDocument" style={{ display: "none" }} onChange={x => this.uploa
dDocumentFile(x.target.files)} />
          <Button onClick={() => this.initializeUploadDocumentFile()} outline color="primary" style={{ wi
dth: "100%" }}>Загрузить</Button>
        </Col>
        : null
      }
     </InputRow>
    </DocInput>

    <DocInput
     visible={doc.placeName !== '' || editing}
     label="Место">
     {this.renderPlace()}
    </DocInput>

    <DocInput
     visible={editing}
```

```
        label="Морфологический анализ">
          {this.renderMorph()}
        </DocInput>

        <DocInput label="">
          <InputRow>
            <Col>
              {
                editing
                  ? <Button outline color="success" style={{ width: "100%" }} disabled={block.length !== 0} onClick={() => this.saveChanges()}>Сохранить документ</Button>
                  : <Button outline color="secondary" style={{ width: "100%" }} onClick={() => this.startEditing()}>Изменить</Button>
              }
            </Col>
          </InputRow>
        </DocInput>
      </div>
    );
  }
}
```