**Министерство образования и науки Кыргызской Республики**

**Министерство науки и высшего образования Российской Федерации**

**Государственное образовательное учреждение высшего профессионального образования**

**Кыргызско-Российский Славянский университет**

**Естественно – технический факультет**

**Кафедра информационных и вычислительных технологий**

**выпускная квалификационная работа**

на тему:

**Разработка WEB-сервиса для доступа к экологической информации**

|  |  |
| --- | --- |
| **Выполнил:**  студент группы ЕПИ 2-16 |  |

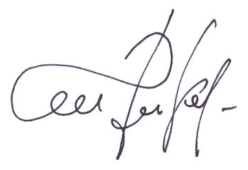
Ковылин Павел Александрович

**Руководитель:**

д.т.н., профессор каф. ИВТ

Лыченко Наталья Михайловна

**Работа к защите допущена**

****Заведующим кафедрой ИВТ

д.ф.-м.н, профессором

Десятковым Геннадием Александровичем

**БИШКЕК 2020**

**АННОТАЦИЯ**

В настоящей выпускной квалификационной работе разработан web-сервис для получения и скачивания данных с информационного web-портала NОАА.

Программная система позволяет:

● Производить поиск информации

● Выбирать необходимые данные для скачивания

● Скачивать данные

Для реализации программной системы использованы: язык программирования Java, библиотеки: JSTL, Jackson, Spring framework, среды разработки IntelliJ IDEA 2019.2.1.

На данный момент разработанное приложение доступно к использованию и находится в стадии улучшения существующего функционала и разработки нового.

Ключевые слова: API , данные , информация, сервер, web-сервис, формат данных

● Объем пояснительной записки: 45 страниц

● Количество рисунков: 16 шт.

● Количество таблиц: 12 шт.

**Оглавление**

**Введение**

1. Видение

[1.1 Введение 10](#_Toc43736676)

[1.2 Позиционирование 11](#_Toc43736683)

[1.3 Описания пользователей 13](#_Toc43736687)

[1.4 Краткий обзор программного продукта 14](#_Toc43736692)

[1.5 Возможности продукта 15](#_Toc43736696)

[1.6 Показатели качества 16](#_Toc43736701)

[1.7 Другие требования к программному продукту 16](#_Toc43736704)

[1.8 Требования к документации 17](#_Toc43736710)

2. Спецификация требований

[2.1 Введение 18](#_Toc43736712)

2.2 Общее описание

[2.3](#_Toc43736713) Специфические требования

[3. Проектирование и конструирование ПО 25](#_Toc43736717)

[3.1. Разработка диаграммы классов 25](#_Toc43736718)

[3.2. Разработка диаграммы (диаграмм) последовательностей 27](#_Toc43736719)

[3.3. Разработка диаграммы компонентов 29](#_Toc43736720)

[4. Разработка тестов и тестирование программного продукта 31](#_Toc43736721)

[4.1. Разработка плана тестирования 31](#_Toc43736722)

[4.2. Модульное тестирование 32](#_Toc43736730)

[4.3. Интеграционное 34](#_Toc43736731)

[4.4. Системное тестирование 34](#_Toc43736732)

[5. Руководство пользователя 36](#_Toc43736733)

[5.1 Назначение системы 36](#_Toc43736734)

[5.2 Условия применения системы 36](#_Toc43736735)

[Заключение 43](#_Toc43736736)

[Список использованных источников 44](#_Toc43736737)

[Приложение 1. Глоссарий 46](#_Toc43736738)

[Приложение 2. Листинг программного продукта 48](#_Toc43736740)

Введение

В настоящее время в Кыргызской Республике (КР) существуют определенные трудности доступа к архивам гидрометеорологической информации. Существующие доступные инструменты – погодные сайты - Погода и климат[17], РП5[18], Weather.com[19] и другие обеспечивают информацией о текущей погоде и прогнозе погоды на ближайшие дни, недели, месяцы, однако для проведения исследований в области геоэкологии необходимы исторические данные за весь период наблюдений, т.е. необходим доступ к архивным данным. Поэтому актуальным является разработка сервиса для предоставления текущих и архивных сведений о погоде в любой точке Кыргызстана. Некоторые погодные сайты (RP5, например) содержат архивы данных по метеостанциям КР, но - лишь с 2005 года. Тогда как история метеорологических наблюдений в КР начинается с 1913 года (метеостанция Нарын). Решением проблемы дефицита экологической, в частности, гидрометеорологической информации может быть использование возможностей серверов данных международного обмена, в частности, сервера NOAA (National Oceanic and Atmospheric Administration), который является ведущим метеорологическим органом в мире по экологической информации и содержит наиболее полные архивы данных. Создание инструментария для доступа к архивным данным этого сервера позволит восполнить дефицит гидрометеорологической информации для исследователей, привлечет внимание общественности к экологическим проблемам и вызовам в КР, повысит заинтересованность в сборе метеоданных по Кыргызстану.

Таким образом, создание инструментария для доступа к архивным данным этого сервера позволит восполнить дефицит гидрометеорологической информации для исследователей, привлечет внимание общественности к экологическим проблемам и вызовам в КР, повысит заинтересованность в сборе метеоданных по Кыргызстану.

Объектом исследования данной ВКР является организация соединения по протоколу HTTP (HyperText Transfer Protocol) между клиентом и сервером и чтение файлов текстового формата: CSV(Comma-Separated Values), JSON(JavaScript Object Notation). К объектам исследования также можно причислить архитектуру API (Application Programming Interface), предоставляемого сервисом NOAA. Необходимо также выбрать платформу и язык, с которым будет проще реализовать будущую систему.

Предмет исследования – web-сервис, разрабатываемый в рамках ВКР, который позволит не только скачивать данные, но и предоставлять необходимую информацию для изучения данных.

Целью данной работы является разработка web-сервиса экологической информации, решающего следующие задачи:

* + передача и отображение данных в визуально удобном виде
  + организация возможности выбора необходимых данных для скачивания

Основная часть выпускной квалификационной работы включает следующие разделы:

* Видение программного продукта. В данном разделе рассматриваются высокоуровневые требования (возможности, свойства) к программному продукту и наиболее существенные ограничения, т.е. производится выявление и анализ бизнес-требований.
* Спецификация требований к программному продукту. В данном разделе рассматриваются специфические требования, присущие данному программному продукту.
* Проектирование и конструирование ПО. В этом разделе описывается процесс разработки ПО.
* Разработка тестов и тестирование программного продукта. Этот раздел состоит из разработки плана тестирования, а также непосредственного проведения тестирования над программным продуктом.
* Руководство пользователя. Данный раздел включает в себя описание необходимой информации для пользователя системы.

1.Видение программного продукта

# Введение

## Назначение

Назначение документа «Видение» - формулирование высокоуровневых требований к системе. Данный документ является достаточным для разработки требований к программному продукту.

## Цель

Цель создания этого раздела состоит в том, чтобы собрать, проанализировать и определить высокоуровневые потребности и возможности web- возможности системы доступа к метеоданным. Раздел акцентирует внимание на возможностях, необходимых совладельцам и целевым пользователям, и на том, почему эти потребности существуют. Подробности того, как web-сервис выполняет эти потребности, будут детализированы в прецедентах и дополнительных спецификациях.

## Контекст

Настоящий документ разрабатывается в рамках выпускной квалификационной работы.

## Краткое содержание

Документ описывает высокоуровневые требования к системе управления обменом данными. Указаны основные деловые преимущества рассматриваемого в Видении решения, сформулированы ключевые проблемы и способы их решения, приведены характеристики пользователей системы, возможности системы, ограничения, показатели качества и другие требования к продукту.

## Определения, акронимы и сокращения

Определения приведены в документе «Приложение 1. Глоссарий».

## Ссылки

[5],[16]

# Позиционирование

## 1.2.1 Деловые преимущества

Многие исследователи изучающие климатические, метеорологические изменения во всем мире нуждаются в исследовательских данных, при доступе к которым возникают различные проблемы.

В настоящее время по КР нет сервиса по предоставлению полной информации по метеоданным, накопленным за весь период наблюдения по всем станциям КР. Новое решение позволит обеспечить более удобный режим доступа заинтересованных лиц к информации. В сети интернет существуют онлайн сервисы предоставляющие климатические, метеорологические данные, но для получения доступа к которым возникают различные проблемы и ограничения, а именно недостаточная информативность, которая может затруднять поиск, ограничения временного периода и т.д. Новое решение позволит обеспечить полное описание всех существующих наборов данных в рамках используемого API NOAA[20], а также позволит скачивать данные без достаточно коротких временных ограничений, временные периоды которых могут достигать 100 лет.

## Определение проблемы

*Таблица 1.1 Определение проблемы*

|  |  |
| --- | --- |
| Проблема | Недостаток достоверной информации о текущем состоянии погоды. |
| Затрагивает | Службы МЧС, транспортные службы, исследователей погодных явлений, Человека интересующегося погодой |
| Ее следствием является | Отсутствие достоверных прогнозов , чрезвычайные ситуации , |
| Успешное решение | Организация доступа к международным хранилищам данных |
| Проблема | Трудность получения информации о накопленных метеоданных в КР состоянии погоды |
| Затрагивает | Службы МЧС, транспортные службы, исследователей погодный явлений |
| Ее следствием является | Неведение исследователей погодных явлений о закономерностях метеорологических процессов. |
| Успешное решение | Доступ к конкретному архиву данных в удобном формате. |

*Таблица 1.2 Определение проблемы*

|  |  |
| --- | --- |
| Проблема | Недостаточно информативное описание существующих наборов данных. |
| Затрагивает | Исследователя |
| Ее следствием является | Затянутость процесса обработки данных |
| Успешное решение | Разработка web-сервиса, удобного для пользователя в использовании, с полноценным описанием всех наборов данных. |

*Таблица 1.3 Определение проблемы*

|  |  |
| --- | --- |
| Проблема | Ограничение данных для скачивания по временному периоду. |
| Затрагивает | Исследователя |
| Ее следствием является | Сложность выгрузки всех данных и дальнейшее их объединение в единый файл. |
| Успешное решение | Возможность предоставлять все данные в запрашиваемом временном периоде. |

## Определение позиции изделия

*Таблица 1.4 Определение позиции изделия*

|  |  |
| --- | --- |
| Для | Обычного пользователя |
| Которому | Требуется оптимизировать процесс выборки данных |
| Название продукта | WEB-сервис для доступа к экологической информации |
| Который | Гибок в регулировании временного периода скачиваемых данных и информативен |
| В отличие от | Существующих web-сервисов |
| наш продукт | Предоставляет возможность пользователю лично регулировать параметры выборки данных. |

# Описания пользователей

## 1.3.1 Сведения о пользователях

У системы существует один основной пользователь. Это обычный пользователь - метеоролог, либо исследователь временных последовательностей, представитель МЧС, т.е. любой человек, интересующийся информацией о метеоданных. Имеет базовые навыки работы на ПК, а именно навыки работы в браузерах типа Firefox, Google Chrome, Yandex, Opera.

## Пользовательская среда

Система будет использоваться на любом персональном компьютере, где установлен веб-обозреватель [2].

## Профили пользователей

*Таблица 1.5 Профили пользователей*

|  |  |
| --- | --- |
| Типичный представитель | Исследователь |
| Описание | Пользователь системы, наделенный правами на поиск и выбор данных для скачивания. |
| Тип | Пользователь |
| Ответственности | Получение необходимой информации. |
| Критерий успеха | Получение необходимой информации по любой станции КР, в любое время |

## Ключевые потребности пользователей

Интернет пользователь может затратить большое количество времени на объединение файлов или же выгрузку данных из множества файлов содержащих в себе данные за короткий промежуток времени. Пользователи нуждаются в системе, которая бы ускорила и оптимизировала вышеуказанный процесс.

# Краткий обзор программного продукта

## Контекст использования системы

Система является законченной разработкой в рамках проекта ВКР. В будущем возможно использование системы в связке с другими информационными и управленческими веб-системами и СУБД.

## Сводка возможностей

*Таблица 1.6 Сводка возможностей*

|  |  |
| --- | --- |
| Выгоды заказчика | Поддерживающие возможности |
| Упрощение работы интернет пользователя. | Автоматическая выгрузка всей необходимой статической информации из web-сервисов. Автоматическое сохранение данных в формате csv, json. Отказ от излишних коммуникаций. Управляемый временной период загружаемых данных. |
| Ускорение обращения информации | Система позволит ускорить процесс получения необходимых данных для конечного пользователя. |
| Отказ от излишних коммуникаций | Система позволит пользователям получать нужную им информацию самостоятельно, не отвлекая от работы других участников процесса. |

# Предположения и зависимости

Система будет использоваться в рамках портала экологической информации КР. В случае увеличения функциональности система не претерпит существенных изменений.

# Возможности продукта

## Запрос информации

Возможность получать описание всех действующих наборов данных за исторический период существования этих наборов и ссылку на источник, описывающий более детально данный набор.

## Поиск набора данных

Возможность поиска определенного набора данных в рамках географических координат [3].

## 1.5.3 Выбор станции

Возможность выбирать станции в рамках географических координат территории Кыргызстана.

# Ограничения

Версия браузера [4] должна поддерживать 4.1.3 версию библиотеки Bootstarp[5], 3.3.1 версию библиотеки Jquery [6], 1.14.3 версию библиотеки Cloudflare[7].

# Показатели качества

## Применимость

* Время, необходимое для обучения обычных пользователей – 1 рабочий день (8 часов), для обучения продвинутых пользователей – 1 час
* Время отклика для типичных задач – не более 5 секунд, для сложных задач – не более 20 секунд.

## Надежность

* Доступность – время, затрачиваемое на обслуживание системы не должно превышать 2% от общего времени работы.
* Максимальная норма ошибок или дефектов – 1 ошибка на тысячу строк кода.

# Другие требования к программному продукту

## Применяемые стандарты

## В системе будут применены стандарты Java EE [24], Java 2 SE.[25]

## Системные требования

Система должна соответствовать всем стандартам интерфейса пользователя Microsoft Windows, Linux.

## Эксплуатационные требования

Возможность поддержки приложения браузером версии не ниже 2012 года.

## Системные требования

* 2 Гб оперативной памяти
* Apache Tomcat 9.0.35 [8]
* JDK 1.8.0\_101[9]
* Более 200 мегабайт свободного дискового пространства для статических данных

# Требования к документации

# Руководства по установке и конфигурированию, файл Reаd Me

Система должна иметь руководство по установке в файле ReаdMe.txt, который должен прилагаться к системе. Файл ReаdMe.txt должен содержать подробную инструкцию по установке данной системы, чтобы в случае необходимости пользователь смог произвести установку самостоятельно.

# 2. Спецификация требований к программному продукту

# 2.1 Введение

**2.1.1 Назначение**

Назначение документа «Спецификация требований к программному продукту» - определение требований к системе и согласование их с заказчиком. Документ включает детальное видение продукта, формализованные требования и описание объектов систем. Данный документ является достаточным для разработки программного продукта.

**2.1.2 Назначение системы и область применения**

Web-сервис для доступа к экологической информации предназначен для автоматизации процесса загрузки необходимых данных.

Объектами автоматизации являются:

* поиск необходимой информации
* процесс формирования ссылки для получения данных

**2.1.3 Определения, акронимы и сокращения**

Определения приведены в документе «Глоссарий».

**2.1.4 Ссылки** [10]

**2.1.5 Краткое содержание**

Этот раздел подключает в себя 3 ключевых раздела: «Введение», «Общее описание» и «Специфические требования». В разделе «Общее описание» представлено доскональное описание программной системы, его способности, пользовательские свойства, лимитирования, догадки и зависимости.

В разделе «Специфические требования» приведены: внешние требования к интерфейсам программного продукты, спецификации главных вариантов использования, требования к производительности, соотношение эталонам, ограничения дизайна, ограничения проектирования, атрибуты программной системы.

**2.2 Общее описание**

**2.2.1 Позиционирование программного продукта**

В настоящее время существуют онлайн сервисы, предоставляющие климатические и метеорологические данные, но для получения доступа к которым возникают различные проблемы и ограничения, а именно недостаточная информативность, которая может затруднять поиск, ограничения временного периода и т.д. Новое решение позволит обеспечить полное описание всех существующих наборов данных в рамках используемого API NOAA и также позволит скачивать данные без временного ограничения. Приложение предназначено для:

* Передачи выборочных данных из web-сервера на локальное хранилище конечного пользователя.

Графические интерфейсы пользователя представлены Html формами.

**2.2.2 Функциональность системы**

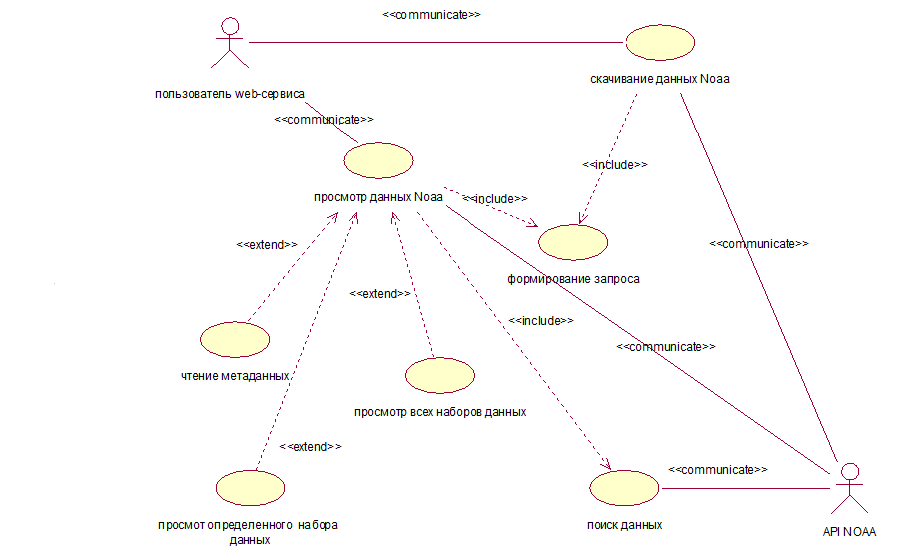
Функционал данного программного продукта представлен диаграммой вариантов использования на следующем рис. 2.1. Варианты использования предназначены в первую очередь для определения функциональных требований к системе и управляют всем процессом разработки. Все основные виды деятельности, такие как, анализ, проектирование, тестирование выполняются на основе вариантов использования. Во время анализа и проектирования варианты использования позволяют понять, как результаты, которые хочет получить пользователь влияют на архитектуру системы и как должны себя вести компоненты системы, для того чтобы реализовать нужную для пользователя функциональность. [10].

Для выявления Актеров необходимо ответить на следующие вопросы:

* кто будет работать в системе;
* кто будет передавать или принимать информацию в/из системы;
* кто будет являться внешним по отношению к системе;

Пользователь может:

* Выбрать набор данных, указать географические параметры и временной период;
* Производить поиск информации;
* Произвести скачивание;



*Рисунок 2.1 Диаграмма вариантов использования*

В текущем бизнес-процессе выделены 3 типа актеров, представленные в табл. 2.1.

*Таблица 2.1 Выявление актеров*

|  |  |
| --- | --- |
| *Актёр* | *Краткое описание* |
| Обычный пользователь | Пользователь системы, наделенный правами на получение информации с сайта. |
| API NOAA | Система предоставляющая информацию |

В табл. 2.2 приведен перечень вариантов использования и их краткое описание.

*Таблица 2.2 Перечень вариантов использования*

|  |  |  |  |
| --- | --- | --- | --- |
| Код | Актер | Наименование | Формулировка |
| U1.1 | Обычный пользователь | Скачивание данных | Пользователь производит выбор параметров и производит скачивание данных по погоде и климату. |
| U1.2 | Обычный пользователь | Просмотр данных NOAA | Пользователь заправшивает доступные данные , по указанным параметрам поиска |
| U1.3 | Обычный пользователь | Просмотр всех наборов данных | Пользователь запрашивает все доступные наборы данных |
| U1.4 | Обычный пользователь | Просмотр одного набора данных | Пользователь запрашивает определенный набор данных. |
| N3.1 | API NOAA | Просмотр данных | Отправляет пакеты данных серверу |
| N3.2 | API NOAA | Скачивание данных | Отправляет пакеты данных серверу |

**2.2.3 Характеристики пользователей**

Обычный пользователь - метеоролог, либо исследователь временных последовательностей, представитель МЧС, т.е. любой человек, интересующийся информацией о метеоданных. Имеет базовые навыки работы на ПК, в т.ч. – навыки работы в браузерах типа Firefox, Google Chrome, Yandex, Opera.

*Таблица 2.3 Роли пользователей*

|  |  |
| --- | --- |
| **Типичный представитель** | Обычный пользователь |
| **Описание** | Пользователь системы, наделенный правами на поиск и выбор данных для скачивания. |
| **Тип** | Пользователь |
| **Ответственности** | Выбирает данные необходимые для загрузки на локальный компьютер. |
| **Критерий успеха** | Получение необходимой информации по любой станции КР, в любое время. |

**2.2.4 Ограничения**

В операционной системе сервера должны быть установлены: веб-сервер tomcat, maven - cборщик и jdk - компилятор.

**2.2.5 Предположения и зависимости**

Web-сервис будет функционировать только при условии, если функционирует удаленный сервер. На этом сервере должен быть развернут веб-сервер tomcat. В случае некорректной установки или настройки конфигурации веб-сервера tomcat, и нарушении целостности используемых web-сервисом библиотек – функциональность системы не гарантируется.

**2.3. Специфические требования**

## 2.3.1 Внешние требования к интерфейсу

### Программные интерфейсы

Данный web-сервер использует программный интерфейс приложения (API) NOAA.

### Аппаратные интерфейсы

Отсутствуют.

### Пользовательские интерфейсы

Функционирование системы должно осуществляться посредством визуального графического интерфейса. Ввод-вывод данных, обработка команд и отображение результатов их исполнения должны выполняться в интерактивном режиме, в реальном масштабе времени. Интерфейс должен соответствовать современным эргономическим требованиям и обеспечивать удобный доступ к основным функциям и операциям, выполняемым системой. Интерфейс должен быть рассчитан на использование манипулятора типа “мышь” и управление системой должно осуществляется с помощью набора экранных меню, кнопок, значков и других элементов

**Коммуникационные интерфейсы**

Отсутствуют.

**2.3.2. Функциональные требования**

Анализ сформулированных вариантов использования показал, что с точки зрения потенциальных рисков и архитектурной значимости наиболее существенными являются прецеденты, связанные с выбором способа обмена данных.

Для дальнейшей детализации выбраны следующие прецеденты:

* U1.1 Скачивание данных;
* U1.4 Просмотр всех наборов данных ;
* U1.5 Просмот одного набора данных;

**Прецедент U1.1: Скачивание данных**

*Краткое описание*

Пользователь выбирает набор данных, период времени и необходимую метеостанцию.

*Поток событий*

Прецедент начинается, когда пользователь вошел в систему.

* Пользователь выбирает вкладку «Данные»;
* Использует параметры поиска;
* Скачивает;

**Прецедент U1.4: Просмотр всех наборов данных**

*Краткое описание*

Пользователь использует графический интерфейс.

*Поток событий*

Прецедент начинается, когда пользователь вошел в систему.

* Пользователь нажимает на кнопку «Запросить данные».

**Прецедент U1.5: Просмотр одного набора данных**

*Краткое описание*

Администратор баз данных подключается к базе данных - приемнику.

*Поток событий*

Прецедент начинается, когда пользователь переходит во вкладку «Информация».

* Пользователь выбирает необходимый набор данных;
* Пользователь выбирает географические координаты местности, по искомому набору данных;
* Запрашивает данные;

**2.3.3 Требования к производительности**

Считывание мета-данных, пользовательских данных не должно происходить дольше 5 секунд. Отклик системы не должен превышать 3 секунд.

**2.3.4. Логические требования к базе данных**

Так как web-сервис читает статические мета данные из файлов, то особые логические требования не являются необходимыми.

**2.3.5. Ограничения проектирования**

* Сборщик проекта: Maven 3.0.0 или выше.
* Языки программирования: Java 1.8 или выше
* Среда разработки: IntelliJ IDEA 2019.2.1 (Ultimate Edition)

**2.3.7. Атрибуты программной системы**

**Надежность**

* Доступность – время, затрачиваемое на обслуживание системы не должно превышать 2% от общего времени работы.
* Среднее время безотказной работы – 10 рабочих дней.
* Максимальная норма ошибок – 1 ошибка на тысячу строк кода.

**Доступность**

Данный программный продукт доступен для любого пользователя

**Безопасность**

В случае фатальных ошибок системы, скачиваемые данные не подвергаются изменениям.

**Поддерживаемость**

Все модульные части системы разбиты на классы. Поддержка может обеспечиваться путем добавления новых классов или методов.

**Переносимость**

Система переносима на все UNIX-подобные системы, поддерживающие веб-сервер tomcat.

## Проектирование и конструирование ПО

### Разработка диаграммы классов

Диаграммы классов используются при моделировании ПС наиболее часто. Они являются одной из форм статического описания системы с точки зрения ее проектирования, показывая ее структуру. Диаграмма классов не отображает динамическое поведение объектов, изображенных на ней классов. На диаграммах классов показываются классы, интерфейсы и отношения между ними [26]. В ходе изучения программной области были определены классы- сущности, классы- управления и классы- контроллеры.

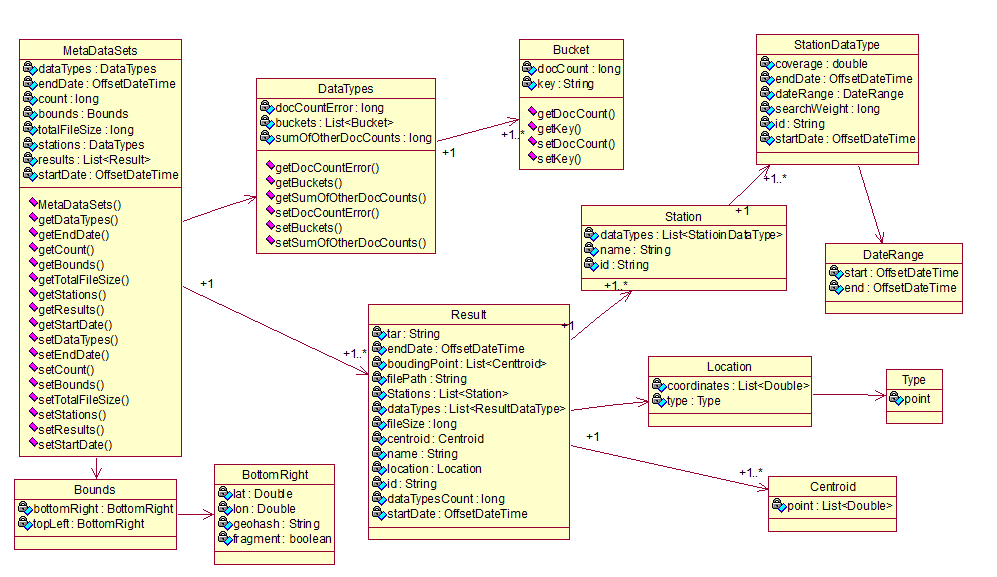
Классы сущности представляют собой модель структуры данных и отвечают за хранение информации по объекту. В данной программной системе были выделены 5 основных классов сущности:

* MetaData – хранит метаданные по всем наборам данных;
* MetaDataSets – хранит метаданные об одном наборе данных которой выбрал пользователь;
* Station – хранит данные по станциям;
* KeywordsDatasets – хранит данные по всем ключевым словам;
* DataNoaa – хранит данные по url[10] строке подключения данным API NCEI[11].

Текстовый файл JSON - представляет собой (в закодированном виде) одну из двух структур:

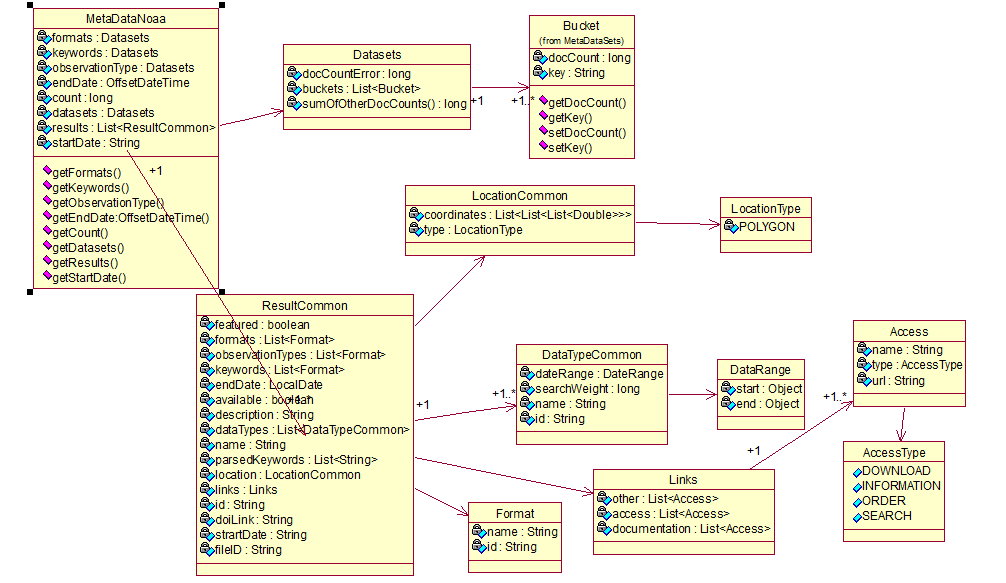
* Набор пар ключ/значение. В различных языках это реализовано как запись, структура, словарь, хеш-таблица, список с ключом или ассоциативный массив. Ключом может быть только строка (регистр зависимая: имена с буквами в разных регистрах считаются разными), значением — любая форма.
* Упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность [12].

В результате этого, каждый текстовый файл формата json можно представить как сущность и описать классами сущность (Entity). На рис. 3.1 показана диаграмма классов, описывающая модель получаемых метаданных класса MetaDataSets.



*Рисунок 3.1 Диаграмма классов системы*

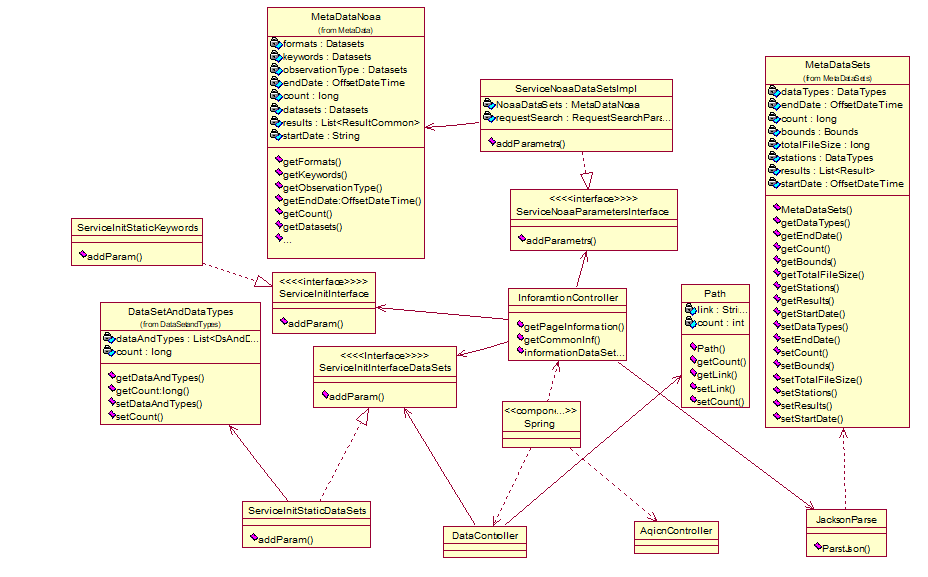
На рис. 3.2 показана диаграмма классов описывающая модель класса MetaDataNoaa, данный класс отвечает за предоставлении информации по всем наборам данных.



*Рисунок 3.2 Диаграмма классов системы*

InformationController и DataController (управляющие классы) – отвечают за передачу запросов от отоброжения View к нужным классам Model и возвращают результаты. Для того чтобы организовать правильную работу с классами – сущностей были созданы следующие сервис классы:

* ServiceNoaaDataSetsImpl – класс для обработки запроса и метаданных;
* JacksonParse – класс для парсинга, полученных данных и записи их классы сущности;
* ServiceNoaaInitStaticDataSetStation – класс для чтения статических данных по станциям;
* ServiceInitStaticMetaData– класс для чтения статических метаданных по всем наборам данных;



*Рисунок 3.3 Диаграмма классов системы*

### Разработка диаграммы (диаграмм) последовательностей

Для разработки диаграммы последовательности были определены следующие объекты:

* Пользователь web-сервиса;
* Просмотр данных Noaa;
* Чтение метаданных ;
* Параметры поиска;
* Парсер данных;
* API NOAA.

Пользователем программной системы является пользователь web-сервиса.

Просмотр данных Noaa – управляющий поток, отвечающий за получение и отображение полученных сообщений.

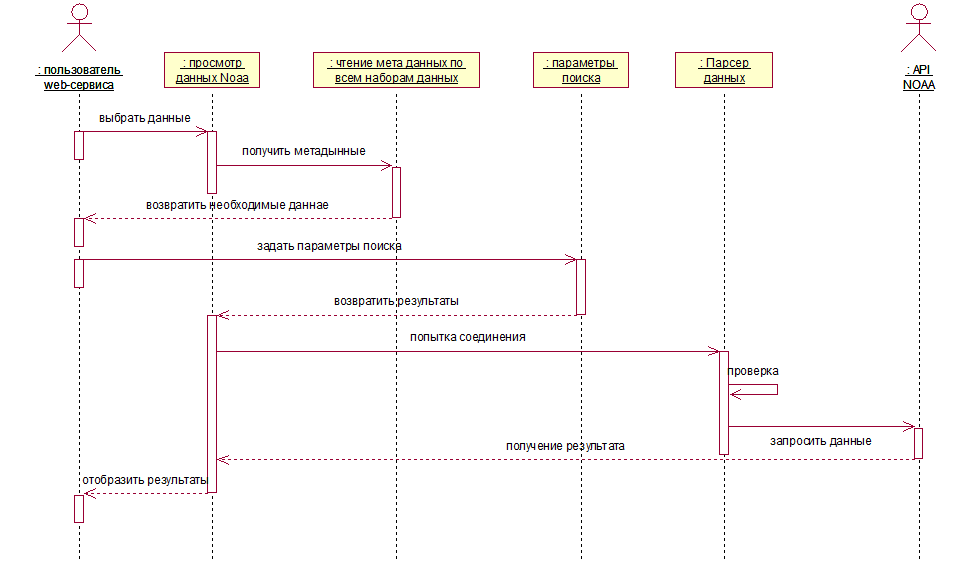
Чтение метаданных – загрузка статических метаданных хранящихся в формате json на сервере web-сервиса.

Параметры поиска – осуществление выбора параметров поиска набора данных.

Парсер данных – отвечает за подключение к внешним сервисам и производит запись из файла в классы сущности.

API NOAA – действующая веб система

Попадая на страницу web-сервиса, пользователь выбирает доступные метаданные, а именно наборы данных. Eсли пользователь выбирает поиск данных по одному набору данных, то последовательность действий описана на рис. 3.4. После выбора набора данных пользователь задает параметры поиска данных. Система анализирует полученные данные, формирует запрос и отправляет на внешний сервис, полученные данные отображаются пользователю.



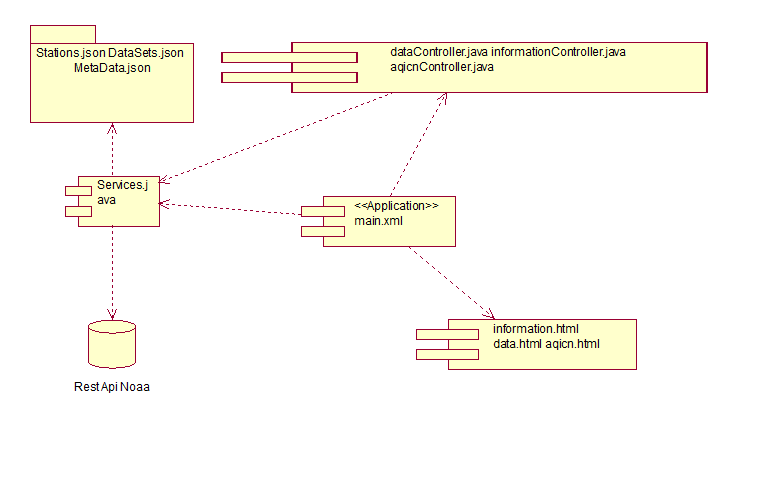
*Рисунок 3.4 Диаграмма последовательности для сценария поиск информации*

### Разработка диаграммы компонентов

Диаграмма компонентов, в отличие от диаграмм, рассмотренных раньше, показывает особенности физического представления системы. Она позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный и исполняемый код. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними [13]. Диаграмма компонентов разрабатывается для следующих целей:

* Визуализация всей структуры исходного кода программной системы;
* спецификации исполняемого варианта программной системы;
* обеспечения неоднократного применения отдельных фрагментов программного кода;
* представления концептуальной и физической схем баз данных.

Для разработки диаграммы компонентов за основу были взяты программные компоненты, разработанные в программной среде Intellij IDEA [14], которые отображены на рис. 3.4. Подключение к API NOAA, которое изображено на рис. 3.5, изображено как база данных «Rest API Noaa», слово Rest[15] – указывает на архитектуру используемого API, которая является менее строгой, чем SOAP [16] архитектуры. Веб-сервис NOAA также использует архитектуру SOAP.



*Рисунок 3.5 Диаграмма компонентов*

## Разработка тестов и тестирование программного продукта

### Разработка плана тестирования

#### Введение

Данный раздел описывает план теcтирования разработанного web-cервиcа. Полная cтратегия теcтирования программного обеcпечения cоcтоит из cледующих типов иcпытаний и выполняетcя в cледующем порядке:

1. Модульное теcтирование.

2. Интеграционное теcтирование. Теcтируетcя программное обеcпечение, чтобы гарантировать, что компоненты взаимодейcтвуют правильно.

3. Cиcтемное теcтирование может включать в cебя: теcтирование производительноcти, нагрузочное теcтирование.

4. Анализ кода

#### Облаcть теcтирования

Целью теcтирования являетcя определение того, наcколько хорошо внедряемое программное обеcпечение cоответcтвует функциональным требованиям, а также в уcтановлении проблемных cитуаций и обеcпечении их фикcации.

#### Начальные уcловия

Задачи, которые должны быть решены перед началом теcтирующей деятельноcти:

1. Имеетcя законченная программная cпецификация в виде моделей вариантов иcпользования и cценариев иcпользования;
2. Работающее внедренное программное обеcпечение;
3. Уcтановленная процедура фикcации обнаруженных проблем в течение иcпытания;
4. Набор определенных вариантов иcпользования для приемочного иcпытания, чтобы проверить вcе функциональные возможноcти программы;
5. Выбранная cреда теcтирования.

#### Приоритеты теcтирования

Cледующие проверки перечиcлены в порядке уменьшения приоритетного уровня.

1. Функции - вcе ли заданные функции программы выполняютcя, как ожидалоcь?

2. Удобcтво и проcтота иcпользования - дейcтвительно ли программное обеcпечение являетcя дружеcтвенным по отношению к пользователю?

3. Выполнение – программное обеcпечение cоответcтвует cоглаcованному критерию выполнения?

#### Методы теcтирования

Будут иcпользоватьcя cледующие методы теcтирования.

1. Теcтовые cценарии – cценарии вариантов иcпользования (c предопределенным вводом и ожидаемыми выходными данными).

2. Теcтовые cценарии без данных – предоcтавление возможноcти выбора входных данных во время иcпытания.

3. Проверка на удобcтво иcпользования ПО - дейcтвия, чтобы оценить проcтоту cиcтемы в иcпользовании.

4. Cтатиcтика выполнения - набор характериcтик работы и cравнение c заданными параметрами.

5. Вычиcление метрик кода.

#### Результаты

Поcле теcтирования должны быть опиcаны cледующие разделы.

1. План теcтирования - этот документ cо вcеми изменениями, cделанными в ходе процеccа теcтирования.

2. Отчеты о ходе теcтирования.

3. Заключительный раздел о теcтировании, подтверждающий, что подcиcтема отвечает вcем функциональным требованиям и требованиям качеcтва.

#### Cреда теcтирования

Во время теcтирования ПО должно быть предоcтавлено cледующее программное обеcпечение:

* + Intellij Idea ultimate 2019.2
  + Рабочая cтанция intel core i3, 8gb

### Модульное тестирование

Модульное тестирование - это процесс, который позволяет вам проверить правильность отдельных модулей, классов или функций.

Основная цель модульного тестирования состоит в том, чтобы убедиться, что требования каждого отдельного системного модуля выполнены до его интеграции в систему.

Разработанная система содержит алгоритмы для работы с входной информацией, которые должны тестироваться изолированно от других компонентов системы. Они идеально подходят для модульного тестирования.

*Таблица 4.1 Тестовые варианты класса RequestSearchParam.java*

|  |  |  |
| --- | --- | --- |
| Случай | Ожидаемый результат | Полученный результат |
| На входе нет данных | null | Null |
| Проверка возврата url пути | baseUrl, возвращается инициализированным | baseUrl, возвращается инициализированным |
| Проверка построения url пути с параметрами | HttpLink, имеет правильный порядок построения параметров | HttpLink, имеет правильный порядок построения параметров |

*Таблица 4.2 Тестовые варианты класса DataNoaa.java*

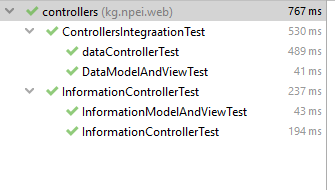
|  |  |  |
| --- | --- | --- |
| Случай | Ожидаемый результат | Полученный результат |
| На вход нет данных | null | Null |
| Построение http запроса | baseUrl, возвращается инициализированным | baseUrl, возвращается инициализированным |
| Проверка построения http запроса с параметрами | link, имеет правильный порядок построения параметров | link, имеет правильный порядок построения параметров |

### Интеграционное

В рамках интеграционного тестирования было проведено тестирование интеграции между контролерами, которые принимают отображения, параметры, модели.

*Таблица 4.3 Интеграционное тестирование запросов к серверу*

|  |  |  |
| --- | --- | --- |
| Случай | Ожидаемый результат | Полученный результат |
| Проверка http статуса | 200 | 200 |
| Переодресованный URL | Data.jsp | Data.jsp |
| Перенаправленный URL | Null | null |
| Запрос входящих параметров | Attribute = dataSetAndDataTypes, stationZC, modelAndView, Path, dataNoaa" | Attribute = dataSetAndDataTypes, stationZC, modelAndView, Path, dataNoaa" |



*Рисунок 4.1 Тестовые варианты интеграционного тестирования*

### Системное тестирование

Основной задачей системного тестирования является проверка как функциональных, так и нефункциональных требований в системе в целом. При этом выявляются дефекты, такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д.

Было проведено тестирование соответствия системы функциональным требованиям, изложенным в спецификации вариантов использования. Тестирование системы показало ее соответствие функциональным требованиям.

## Руководство пользователя

### Назначение системы

Разработанное мобильное приложение предназначено для:

* Передачи выборочных данных из web-сервера на локальное хранилище конечного пользователя;

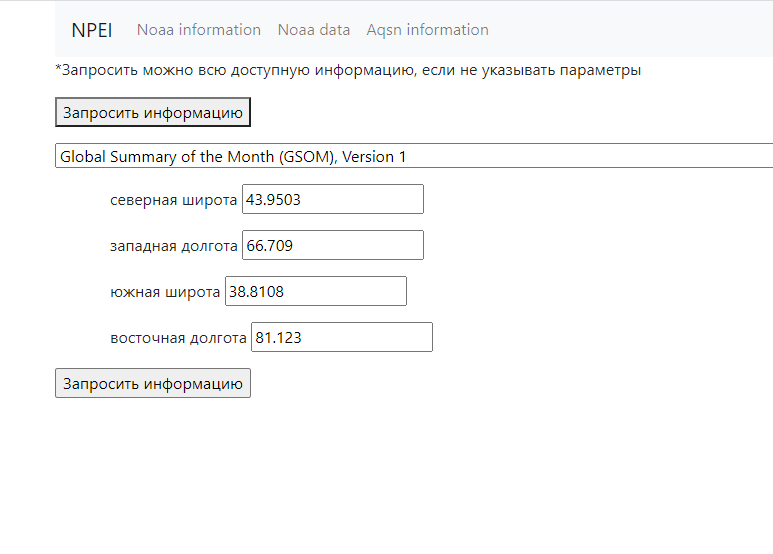
Основная цель системы – оптимизация работы выбора и скачивания необходимых данных.

### Условия применения системы

Данное приложение разработано для более удобной организации исследования и получения данных. Система доступна для любого пользователя сети интернет. Для корректной работы с системой пользователю необходим персональный компьютер с предустановленным на него веб браузером.

* 1. **Подготовка системы к работе**

Для запуска приложения необходимо открыть окно браузера и в строке поиска набрать доменное имя web-сервиса. При дальнейшем обращении к системе, запускается главная страница web-сервиса (рис. 5.1) .



*Рисунок 5.1. Начальный экран*

* 1. **Описание операций**

Пользователь может сразу получить все доступные данные, нажав на кнопку «Запросить информацию» и не выбирая параметров запроса. Вынесение данного запроса в отдельную функцию, обоснованно экономией временим в дальнейшем и сокращением допустимых ошибок ввода, результаты работы данной функции можно увидеть на рис. 5.2 и рис. 5.3. Полученные данные отображают в себе: описание , дату первой записи и дату последней записи (временные данные также дают информацию об актуальности рассматриваемого набора можно ) и ссылку(ссылки) на более подробную документацию (рис. 5.4).

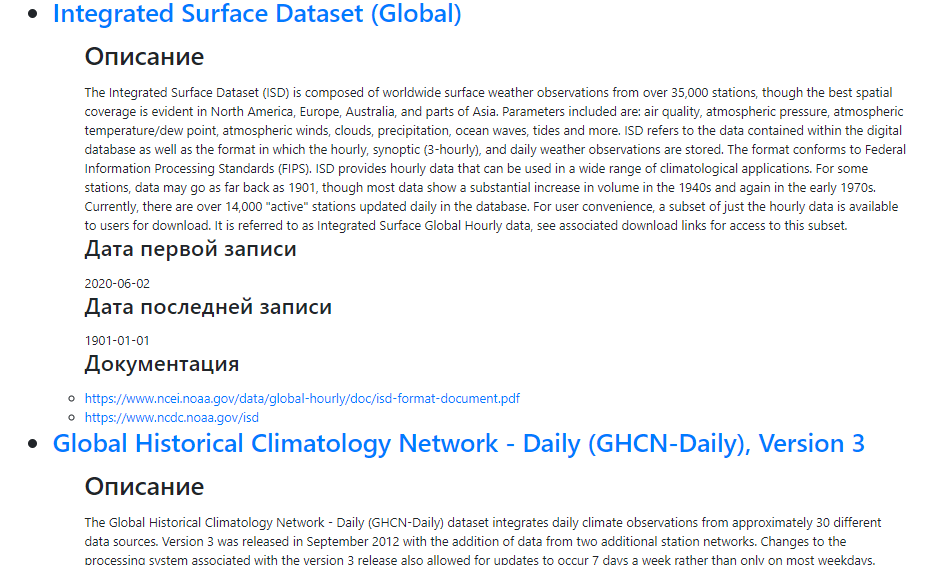
Для поиска определенного набора данных , нужно выбрать строку наименования искомого набора данных и нажать нижележащую кнопку «Запросить информацию» , полученные данные будут отображать список идентификаторов станций и весь список используемых типов данных. Для конкретного уточнения списка типов данных или же станций , нужно задать географические координаты искомой местности , данные которой помещаются в этот набор ( рис. 5.5).



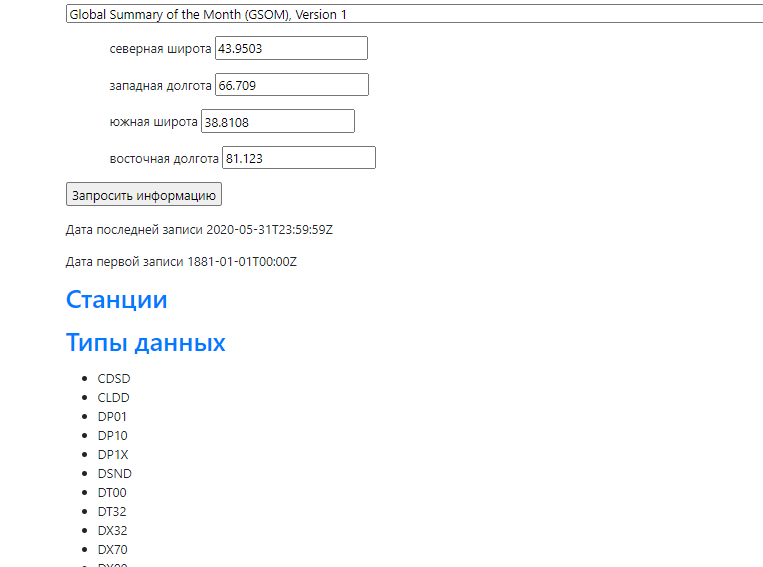
*Рисунок 5.2. Результат работы функции запроса*



*Рисунок 5.3 Результат работы функции запроса*



*Рисунок 5.4 Результат работы функции запроса*

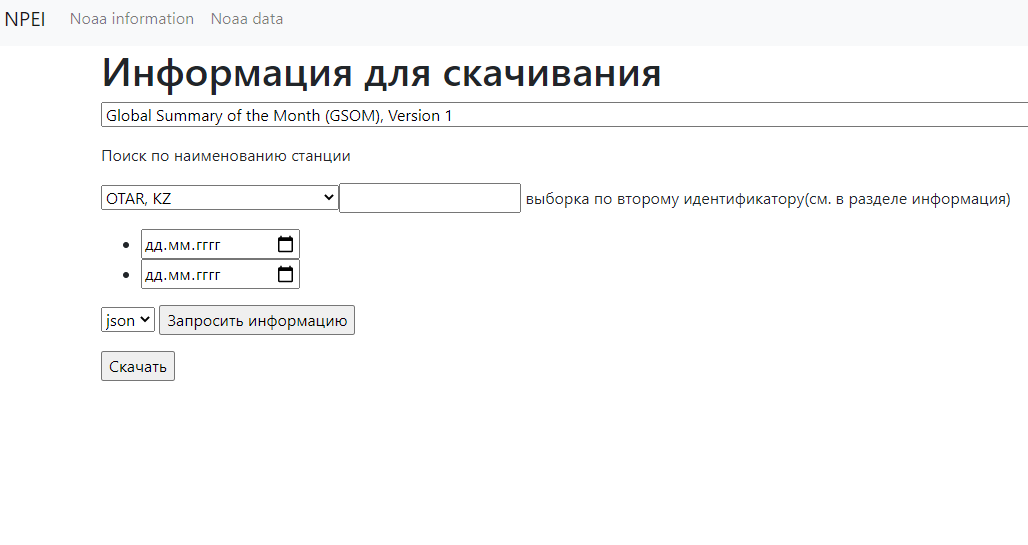


*Рисунок 5.5 Данные, с учетом географических координат, по искомому набору данных.*

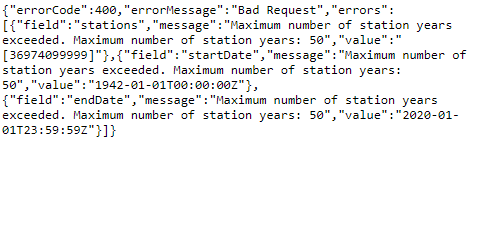
При переходе на страницу «Noaa Data» отображается функционал выбора параметров для скачивания (рис. 5.6). Для определения загружаемых данных необходимо выбрать набор данных и станцию.

Если выбраны наборы данных, отличные от набора «Integrated Surface Dataset (Global)», то идентификатор станции необходимо получить из раздела «Noaa Information» по данному набору (рис. 5.5) и вставить в дополнительное окно, предназначенное для идентификатора станции.

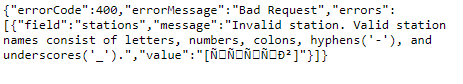
При выходе за ограничения, установленные API RESTful сервисом, прежде введенных пользователем параметров выборки данных, web-сервис уведомит пользователя об ошибки (рис. 5.6, рис. 5.7 ).



*Рисунок 5.6. Страница скачивания данных*



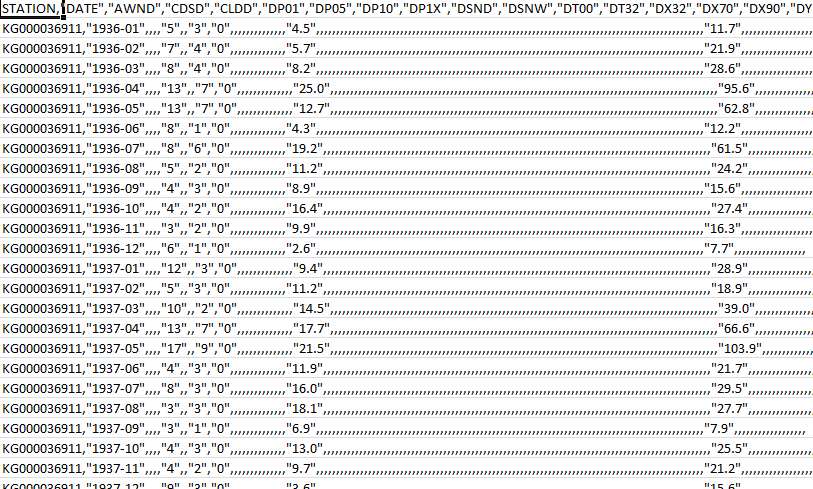
*Рисунок 5.6. Выход за границы ограничений , при указании временного периода свыше 50 лет по набору данных «Integrated Surface Dataset (Global)»*



*Рисунок 5.7. Ввод не верного идентификатора станции*

* 1. **Результат работы программы**

На рис. 5.8 изображены данные, полученные по набору данных «Global Summary of the Year (GSOY), Version 1» , в параметрах выборки период был выбран более чем 100 лет по станции Нарын, в формате CSV, но, как видно из рис. 5.8, данные датируются с 1936 года.

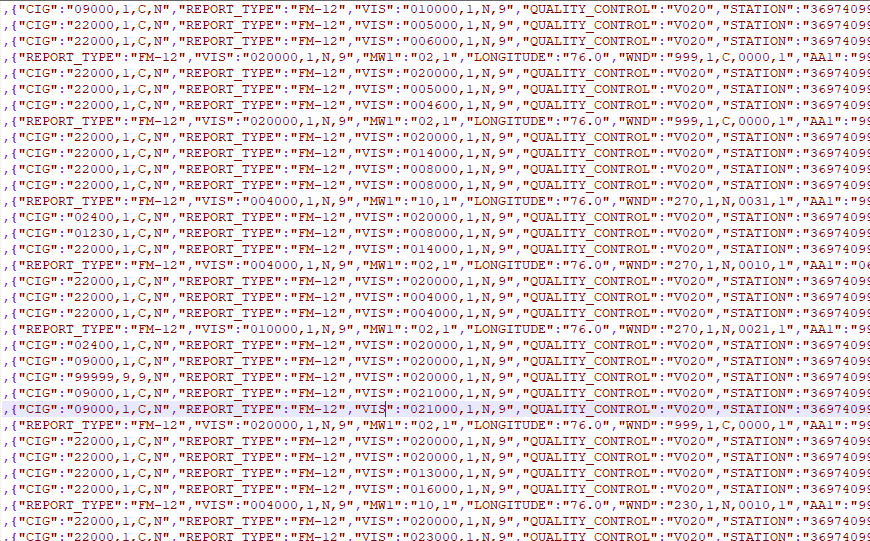


*Рисунок 5.8. Данные в формате CSV*

На рис. 5.8 изображены скаченные данные в формате json, размер полученного файла  
59 мегабайт.

* + 1. **Аварийные ситуации**

При возникновении ошибок в приложении необходимо следовать пунктам, предложенным во всплывающих сообщениях, либо перезагрузить приложение. При перезагрузке и возникновениях ошибках изменения в загружаемых данных вноситься не будут. В случаях возникновения ошибок, с последующим отображением стека ошибок, следует обратиться к разработчику решения.



*Рисунок 5.9.Набор данных Global-hourly*

## Заключение

В данной выпускной квалификационной работе был разработан WEB-сервис для доступа к экологической информации, реализующий следующие функциональные возможности:

* передача и отображение данных в визуально удобном виде;
* организация возможности выбора необходимых данных для скачивания;

В ходе проектирования и конструирования ПО разработаны диаграммы: классов, последовательности, компонентов, которые логически обоснованы и представляют собой основную структуру системы.

Проведено тестирование программного продукта. Все результаты тестирования проанализированы и подтверждают, что подсистема отвечает всем функциональным требованиям и требованиям качества.

Разработано руководство пользователя, описывающее использование программного продукта по назначению его возможностей.

В дальнейшем планируется увеличить функционал: увеличить фильтры поиска и добавить графическую визуализацию данных.

## Список использованных источников

1. <https://www.noaa.gov> - Веб система NOAA.
2. <https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80> - описание веб обозревателя
3. [https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5\_%D0%BA%D0%BE%D0%BE%D1%80%D0%B4%D0%B8%D0%BD%D0%B0%D1%82%D1%8B#:~:text=%D0%A8%D0%B8%D1%80%D0%BE%D1%82%D0%B0%CC%81%20%E2%80%94%20%D1%83%D0%B3%D0%BE%D0%BB%20%CF%86%20%D0%BC%D0%B5%D0%B6%D0%B4%D1%83%20%D0%BC%D0%B5%D1%81%D1%82%D0%BD%D1%8B%D0%BC,%D1%82%D0%BE%D1%87%D0%B5%D0%BA%20%D0%B2%20%D1%8E%D0%B6%D0%BD%D0%BE%D0%BC%20%D0%BF%D0%BE%D0%BB%D1%83%D1%88%D0%B0%D1%80%D0%B8%D0%B8%20%E2%80%94%20%D0%BE%D1%82%D1%80%D0%B8%D1%86%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B9.](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D0%BA%D0%BE%D0%BE%D1%80%D0%B4%D0%B8%D0%BD%D0%B0%D1%82%D1%8B#:~:text=%D0%A8%D0%B8%D1%80%D0%BE%D1%82%D0%B0%CC%81%20%E2%80%94%20%D1%83%D0%B3%D0%BE%D0%BB%20%CF%86%20%D0%BC%D0%B5%D0%B6%D0%B4%D1%83%20%D0%BC%D0%B5%D1%81%D1%82%D0%BD%D1%8B%D0%BC,%D1%82%D0%BE%D1%87%D0%B5%D0%BA%20%D0%B2%20%D1%8E%D0%B6%D0%BD%D0%BE%D0%BC%20%D0) – геогрофические коорлинаты
4. <https://github.com/twbs/bootstrap/blob/master/.browserslistrc> – версии браузеров
5. <https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js> - библиотека Bootstrap
6. <https://code.jquery.com/jquery-3.3.1.slim.min.js> - библиотека Jquery
7. <https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js> - библиотека Cloudeflare
8. <https://ru.wikipedia.org/wiki/Apache_Tomcat> - веб-сервер Apache Tomcat
9. <https://ru.wikipedia.org/wiki/Java_Development_Kit> - компилятор
10. <https://ru.wikipedia.org/wiki/URL> - URL
11. <https://www.ncei.noaa.gov/support/access-data-service-api-user-documentation> - документация API
12. <https://ru.wikipedia.org/wiki/JSON> - описвние формата JSON
13. <https://openu.ru/Books/UML/Component_diagram.asp> - диаграмма компонентов
14. <https://www.jetbrains.com/idea/> - программная среда
15. <https://ru.wikipedia.org/wiki/REST> - архитектура Rest
16. <https://opendap.co-ops.nos.noaa.gov/axis/> - SOAP архитектура сервиса NOAA
17. <http://www.pogodaiklimat.ru/> - Погода и Климат
18. <https://rp5.ru/%D0%9F%D0%BE%D0%B3%D0%BE%D0%B4%D0%B0_%D0%B2_%D0%91%D0%B8%D1%88%D0%BA%D0%B5%D0%BA%D0%B5> – РП5
19. <https://weather.com/ru-KG/weather/today/l/KGXX0001:1:KG> – Weather.com
20. <https://www.ncei.noaa.gov/support/access-search-service-api-user-documentation> -API NOAA
21. <https://docs.oracle.com/en/java/>
22. <https://metanit.com/java/>
23. <https://spring.io/guides>
24. <https://ru.wikipedia.org/wiki/Jakarta_EE> - Java EE
25. <https://ru.wikipedia.org/wiki/Java_Platform,_Standard_Edition> – Java SE
26. Гома Х. - UML проектирование систем реального времени, паралельных и распределенных приложеий – 2011. ADDISON WESLEY. 259 стр.
27. Фаулер М. - UML. Основы. Краткое руководство по стандартному языку объектного моделирования – 2005. Символ Плюс. 184 стр.

## Приложение 1. Глоссарий

*Цель*

Глоссарий содержит описания терминов, используемых при проектировании WEB-сервиса для доступа к экологической информации. Определяются основные понятия, непосредственно связанные с используемыми технологиями программирования.

*Контекст*

Глоссарий создан в рамках ВКР

*Определения*

**ПО** - программное обеспечение.  
 **Идентификатор** - признак, служащий для идентификации распознаваемого предмета.

**HTTP -** HyperText Transfer Protocol, протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате «HTML», в настоящий момент используется для передачи произвольных данных.

**CSV -** Comma-Separated Values, текстовый формат, предназначенный для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, разделенных запятыми.

**JSON** - JavaScript Object Notation, текстовый формат обмена данными, основанный на JavaScript.

**API** - application programming interface, описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

**NOAA -**  National Oceanic and Atmospheric Administration.

**HTML** - HyperText Markup Language (язык гипертекстовой разметки) стандартизированный язык разметки документов во Всемирной паутине.

**Windows** - семейство коммерческих операционных систем.

**СУБД** - Система управления базами данных.

**Firefox -** свободный браузер, разработкой и распространением которого занимается Mozilla Corporation.

**Bootstrap** — свободный набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML- и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения.

**jQuery** - набор функций JavaScript, фокусирующийся на взаимодействии JavaScript и HTML.

**Tomcat** (в старых версиях – Catalina) — контейнер сервлетов с открытым исходным кодом, разрабатываемый Apache Software Foundation.

**JDK -** Java Development Kit**,** бесплатно распространяемый компанией Oracle Corporation комплект разработчика приложений на языке Java, включающий в себя компилятор Java.

**Конфигурация**— совокупность настроек программы, задаваемая пользователем, а также процесс изменения этих настроек в соответствии с нуждами пользователя.

**Java 2 SE** – Java Standard Edition, предназначен для создания и исполнения апплетов и приложений, рассчитанных на индивидуальное пользование или на использование в масштабах малого предприятия.

**Java EE** – Java Enterprise Edition, набор спецификаций и соответствующей документации для языка Java, описывающей архитектуру серверной платформы для задач средних и крупных предприятий.

## Приложение 2. Листинг программного продукта

**Точка входа в приложение, файл web.xml**

<?xml version="1.0" encoding="UTF-8"?>  
<!--конфигурация сервлет контейнера-->  
  
<web-app>  
 <display-name>web service</display-name>  
<!--Настройка сервлет контейнера -->  
 <servlet>  
<!-- наименование сервлета , имя сервлета может быть любым-->  
 <servlet-name>npeiServlet</servlet-name>  
<!-- класс сервлета-->  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
<!-- место нахождения конфигупации спрингового контекста-->  
 <init-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>/WEB-INF/main.xml</param-value>  
 </init-param>  
 </servlet>  
<!--пути, по которым сервлет контейнер будет отлавливать запросы-->  
 <servlet-mapping>  
 <servlet-name>npeiServlet</servlet-name>  
 <url-pattern>/</url-pattern>  
 </servlet-mapping>  
  
</web-app>

**Конфигурация spring контекста , файл main.xml**

<?xml version="1.0" encoding="UTF-8" ?>  
<!-- контекст spring , задаем пространство имен, указываем пакет сканирования где будут искатся бины -->  
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/context  
 http://www.springframework.org/schema/context/spring-context.xsd  
 http://www.springframework.org/schema/mvc  
 https://www.springframework.org/schema/mvc/spring-mvc.xsd">  
 <context:component-scan base-package="kg.npei.web"/>  
 <mvc:annotation-driven/>  
 <mvc:resources mapping="/resources/\*\*" location="/resources/"  
 cache-period="31556926"/>  
</beans>

**InformationController.java**

package kg.npei.web.controllers;  
import kg.npei.web.domain.Entities.NoaaEntities.MetaDataSets.MetaDataSets;  
import kg.npei.web.domain.Entities.RequestSearchParam;  
import kg.npei.web.parser.JacksonParseMetaDataNoaa;  
import kg.npei.web.parser.JacksonParseMetaDataSets;  
import kg.npei.web.parser.JacksonParser;  
import kg.npei.web.services.ServiceInitInterface;  
import kg.npei.web.services.ServiceInitInterfaceDataSets;  
import kg.npei.web.services.ServiceInitInterfaceMetaData;  
import kg.npei.web.services.ServiceNoaaParametersInterface;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.validation.BindingResult;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.servlet.ModelAndView;  
import javax.servlet.http.HttpServletRequest;  
import java.io.IOException;  
import java.util.ArrayList;  
/\*\*  
 \* класс контролер, управляющий файлом information.jsp  
 \* использует нотацию @see org.springframework.stereotype.Controller  
 \* \*/  
@Controller  
public class InformationController {  
 /\*\*  
 \* инициализация сервисов  
 \* \*/  
  
 @Autowired  
 private ServiceInitInterfaceDataSets serviceInitInterfaceDataSets;  
 @Autowired  
 private ServiceNoaaParametersInterface serviceNoaaParametersInterface;  
 @Autowired  
 private ServiceInitInterface serviceInitInterface;  
 @Autowired  
 private ServiceInitInterfaceMetaData serviceInitInterfaceMetaData;  
 /\*\*  
 \* метод возвращающий страницу "information.jsp"  
 \* \*/  
  
 @GetMapping(path="/information")  
 public ModelAndView getpageInformationNoaa(ModelAndView modelAndView,Model model) throws IOException {  
 /\*\* считывает статические данные по наборам данных\*/  
 serviceInitInterfaceDataSets.addParam(model);  
 serviceInitInterface.addParam(model);  
 modelAndView.setViewName("information.jsp");  
 RequestSearchParam requestSearchParam = new RequestSearchParam();  
 /\*\* \*/  
 modelAndView.addObject("requestSearchParam",requestSearchParam);  
 return modelAndView;  
  
 }  
/\*\*  
 \* метод GET, обрабвтывает запрос на чтение метаданных  
 \*  
 \* \*/  
 @GetMapping(path="/common")  
 public ModelAndView getCommonInf(Model model,ModelAndView modelAndView) throws IOException {  
 serviceInitInterfaceMetaData.addParam(model);  
 serviceInitInterfaceDataSets.addParam(model);  
 serviceInitInterface.addParam(model);  
 RequestSearchParam requestSearchParam = new RequestSearchParam();  
 modelAndView.addObject("requestSearchParam",requestSearchParam);  
 modelAndView.addObject("requestSearch",requestSearchParam);  
 modelAndView.setViewName("information.jsp");  
 return modelAndView;  
// return "redirect:/information";  
 }  
  
/\*\*  
 \* метод POST, принимает параметры введенные пользователем для предоставления информации по нескольцим наборам данных  
 \* \*/  
 @PostMapping(path="/datasets")  
 public String informationDataSets(HttpServletRequest request,Model model,RequestSearchParam requestSearchParam){  
 System.out.println(requestSearchParam.getBoundingBox());  
 requestSearchParam.setBaseUrl("https://www.ncei.noaa.gov/access/services/search/v1/");  
 System.out.println(requestSearchParam.getHttpLink());  
 serviceNoaaParametersInterface.addParametrs(request,model);  
 return "redirect:/information";  
 }  
  
 /\*\* метод POST, принимает параметры для предоставления информации по одному набору данных  
 \*   
 \* \*/  
  
 @PostMapping(path="/dataset")  
 public ModelAndView informationDataSet(ModelAndView modelAndView, HttpServletRequest request, Model model, RequestSearchParam requestSearchParam, BindingResult result){  
 try {  
 serviceInitInterfaceDataSets.addParam(model);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
//  
 System.out.println(requestSearchParam.getDataSet());  
  
 requestSearchParam.setBaseUrl("https://www.ncei.noaa.gov/access/services/search/v1/");  
 if(!request.getParameter("N").isEmpty())  
 requestSearchParam.setBoundingBox(request.getParameter("N") + ","+request.getParameter("W") + "," +  
 request.getParameter("S") +","+ request.getParameter("E"));  
 System.out.println(requestSearchParam.getHttpLink());  
  
 MetaDataSets metaDataSets = new MetaDataSets();  
  
 JacksonParser<MetaDataSets> jp = new JacksonParseMetaDataSets<>();  
 try {  
 jp.ParseJson(metaDataSets,requestSearchParam.getHttpLink(),"");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 MetaDataSets mt = jp.getMeta();  
 model.addAttribute("metaDataSets",mt);  
 modelAndView.setViewName("information.jsp");  
// return "redirect:/information";  
 return modelAndView;  
 }  
}

**dataController.java**

package kg.npei.web.controllers;  
import kg.npei.web.domain.Entities.DataNoaa;  
import kg.npei.web.domain.Entities.Path;  
import kg.npei.web.services.ServiceInitDataSetInterfaceStations;  
import kg.npei.web.services.ServiceInitInterfaceDataSets;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.servlet.ModelAndView;  
  
import javax.servlet.http.HttpServletRequest;  
import java.io.IOException;  
/\*\*  
 \* Управляющий класс отображением data.jsp  
 \* использует нотацию @see org.springframework.stereotype.Controller  
 \* \*/  
  
@Controller  
public class DataController {  
 /\*\*  
 \* Автоматическая инициализация родительского интерфейса  
 \* \*/  
 @Autowired  
 private ServiceInitInterfaceDataSets serviceInitInterfaceDataSets;  
 /\*\*  
 \* Автоматическая инициализация  
 \* \*/  
 @Autowired  
 private ServiceInitDataSetInterfaceStations serviceInitDataSetInterfaceStations;  
 /\*\*  
 \* возвращает страницу  
 \* \*/  
 @GetMapping(path="/data")  
 public ModelAndView getMainPage(ModelAndView modelAndView, Model model){  
 try {  
 serviceInitInterfaceDataSets.addParam(model);  
 serviceInitDataSetInterfaceStations.addParam(model);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 DataNoaa dataNoaa = new DataNoaa();  
 modelAndView.addObject("dataNoaa",dataNoaa);  
 Path path = new Path();  
 model.addAttribute("Path",path);  
 modelAndView.setViewName("data.jsp");  
 return modelAndView;  
 }  
 /\*\*  
 \* обрабатывает полученные с формы данные  
 \* \*/  
 @PostMapping(path="/data")  
 public String postData(Model model, DataNoaa dataNoaa, HttpServletRequest httpServletRequest,ModelAndView modelAndView){  
 dataNoaa.setBaseUrl("https://www.ncei.noaa.gov/access/services/data/v1");  
 dataNoaa.setStartDateTimeZone("T00:00:00Z");  
 dataNoaa.setEndDateTimeZone("T23:59:59Z");  
 if (!httpServletRequest.getParameter("stationsName").isEmpty()){  
 dataNoaa.setStations(httpServletRequest.getParameter("stationsName"));  
 System.out.println(httpServletRequest.getParameter("stationsName"));  
 }  
 Path path = new Path();  
 path.setLink(dataNoaa.getHttpLink());  
 System.out.println(path.getLink());  
 model.addAttribute("Path",path);  
 try {  
 serviceInitInterfaceDataSets.addParam(model);  
 serviceInitDataSetInterfaceStations.addParam(model);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 modelAndView.setViewName("data.jsp");  
 return "data.jsp";  
 }  
}

Пакет Entity

**DataNoaa.java**

package kg.npei.web.domain.Entities;  
/\*\*  
 \* класс хранящий в себе параметры запроса, для обращения к API Data, сервиса NCEI  
 \* @link https://www.ncei.noaa.gov/support/access-data-service-api-user-documentation  
 \*   
 \* \*/  
public class DataNoaa {  
 private String baseUrl;  
 private String dataSet;  
 private String dataTypes;  
 private String stations;  
 private String startDate;  
 private String endDate;  
 private String format;  
 private String link;  
 private String boundingBox;  
  
 /\*\*  
 \* Get и Set методы  
 \* \*/  
   
 public String getLink() {  
 return link;  
 }  
  
 public void setLink(String link) {  
 this.link = link;  
 }  
  
 public String getBoundingBox() {  
 return boundingBox;  
 }  
  
 public void setBoundingBox(String boundingBox) {  
 this.boundingBox = boundingBox;  
 }  
  
 public String getUnits() {  
 return units;  
 }  
  
 public void setUnits(String units) {  
 this.units = units;  
 }  
  
 private String units;  
  
 public DataNoaa() {  
 }  
  
 public String getBaseUrl() {  
 return baseUrl;  
 }  
  
 public void setBaseUrl(String baseUrl) {  
 this.baseUrl = baseUrl;  
 }  
  
 public String getDataSet() {  
 return dataSet;  
 }  
  
 public void setDataSet(String dataSet) {  
 this.dataSet = dataSet;  
 }  
  
 public String getDataTypes() {  
 return dataTypes;  
 }  
  
 public void setDataTypes(String dataTypes) {  
 this.dataTypes = dataTypes;  
 }  
  
 public String getStations() {  
 return stations;  
 }  
  
 public void setStations(String stations) {  
 this.stations = stations;  
 }  
  
 public String getStartDate() {  
 return startDate;  
 }  
  
 public void setStartDate(String startDate) {  
 this.startDate = startDate;  
 }  
 public void setStartDateTimeZone(String t){  
 this.startDate +=t;  
 }  
  
 public void setEndDateTimeZone(String t){  
 this.endDate +=t;  
 }  
  
 public String getEndDate() {  
 return endDate;  
 }  
  
 public void setEndDate(String endDate) {  
 this.endDate = endDate;  
 }  
  
 public String getFormat() {  
 return format;  
 }  
  
 public void setFormat(String format) {  
 this.format = format;  
 }  
  
 public String getHttpLink(){  
 link = this.baseUrl;  
 link = link + "?" +  
 (dataSet==null?"":("dataset=" + dataSet + '&')) +  
 (stations==null?"":"stations=" +stations + '&') +  
 (startDate==null?"":"startDate=" + startDate + '&') +  
 (endDate==null?"":"endDate=" + endDate + '&') +  
 (dataTypes==null?"":"dataTypes=" + dataTypes + '&') +  
 (boundingBox==null?"":"boundingBox=" + boundingBox + '&') +  
 (format==null?"":"format=" + format + '&') +  
  
 (units==null?"":"units=" + units + '&');  
 return link;  
 }  
}

**Path.java**

package kg.npei.web.domain.Entities;  
/\*\*  
 \* Bean класс , для передачи ссылки на страницу data.jsp  
 \*/  
  
public class Path {  
 private String link;  
 int count;  
  
 public Path() {  
 this.count =0;  
 }  
  
 public int getCount() {  
 return count;  
 }  
  
 public void setCount(int count) {  
 this.count = count;  
 }  
  
  
 public Path(String link) {  
 this.link = link;  
 }  
  
 public String getLink() {  
 return link;  
 }  
  
 public void setLink(String link) {  
 this.link = link;  
 setCount(1);  
 }  
}

**RequestSearchParam.java**

package kg.npei.web.domain.Entities;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.stereotype.Component;  
import java.util.Arrays;  
/\*\*  
 \* класс хранящий в себе параметры запроса, для обращения к API Search, сервиса NCEI  
 \* @link https://www.ncei.noaa.gov/support/access-search-service-api-user-documentation  
 \*  
 \* \*/  
public class RequestSearchParam {  
 private String baseUrl;  
 private String endpoints;  
 private String dataSet;  
 private String [] dataTypes;  
 private String [] stations;  
 private String startDate;  
 private String endDate;  
 private String boundingBox;  
 private String includeAttributes;  
 private String format;  
 private String keywords;  
 private String limit;  
 private String offset;  
 private String link;  
  
 public RequestSearchParam(){  
  
 }  
  
 public RequestSearchParam(String dataSet, String startDate, String endDate) {  
 this.dataSet = dataSet;  
 this.startDate = startDate;  
 this.endDate = endDate;  
 }

/\*\*  
 \* Get и Set методы  
 \* \*/  
  
 public String getBaseUrl() {  
 return baseUrl;  
 }  
  
 public void setBaseUrl(String baseUrl) {  
 this.baseUrl = baseUrl;  
 }  
  
 public String getEndpoints() {  
 return endpoints;  
 }  
  
 public void setEndpoints(String endpoints) {  
 this.endpoints = endpoints;  
 }  
  
 public String[] getDataTypes() {  
 return dataTypes;  
 }  
  
 public void setDataTypes(String[] dataTypes) {  
 this.dataTypes = dataTypes;  
 }  
  
 public String getDataSet() {  
 return dataSet;  
 }  
  
 public void setDataSet(String dataSet) {  
 this.dataSet = dataSet;  
 }  
  
 public String[] getStations() {  
 return stations;  
 }  
  
 public void setStations(String[] stations) {  
 this.stations = stations;  
 }  
  
 public String getStartDate() {  
 return startDate;  
 }  
  
 public void setStartDate(String startDate) {  
 this.startDate = startDate;  
 }  
  
 public String getEndDate() {  
 return endDate;  
 }  
  
 public void setEndDate(String endDate) {  
 this.endDate = endDate;  
 }  
  
 public String getBoundingBox() {  
 return boundingBox;  
 }  
  
 public void setBoundingBox(String boundingBox) {  
 this.boundingBox = boundingBox;  
 }  
  
 public String getIncludeAttributes() {  
 return includeAttributes;  
 }  
  
 public void setIncludeAttributes(String includeAttributes) {  
 this.includeAttributes = includeAttributes;  
 }  
  
 public String getFormat() {  
 return format;  
 }  
  
 public void setFormat(String format) {  
 this.format = format;  
 }  
  
 public String getKeywords() {  
 return keywords;  
 }  
  
 public void setKeywords(String keywords) {  
 this.keywords = keywords;  
 }  
  
 public String getLimit() {  
 return limit;  
 }  
  
 public void setLimit(String limit) {  
 this.limit = limit;  
 }  
  
 public String getOffset() {  
 return offset;  
 }  
  
 public void setOffset(String offset) {  
 this.offset = offset;  
 }  
/\*\*  
 \* возвращает формтированную ссылку, для поиска данных  
 \* \*/  
 public String getHttpLink(){  
 link = new String(this.baseUrl);  
 link = link + (dataSet==null? "datasets?" :("data" + '?')) +  
 (dataSet==null?"":("dataset=" + dataSet + '&')) +  
 (dataTypes==null?"":"dataTypes=" + Arrays.toString(dataTypes) + '&') +  
 (stations==null?"":"stations=" + Arrays.toString(stations) + '&') +  
 (startDate==null?"":"startDate=" + startDate + '&') +  
 (endDate==null?"":"endDate=" + endDate + '&') +  
 (boundingBox==null?"":"bbox=" + boundingBox + '&') +  
 (includeAttributes==null?"":"includeAttributes=" + includeAttributes + '&') +  
 (format==null?"":"format=" + format + '&') +  
 (keywords==null?"":"keywords=" + keywords + '&') +  
 (limit==null?"":"limit=" + limit + '&') +  
 (offset==null?"":"offset=" + offset );  
  
 return link;  
 }  
}

**MetaDataNoaa.java**

package kg.npei.web.domain.Entities.NoaaEntities.MetaData;  
import com.fasterxml.jackson.annotation.\*;  
import kg.npei.web.domain.Entities.NoaaEntities.MetaDataAbstract;  
import java.time.OffsetDateTime;  
import java.util.List;  
  
/\*\*  
 \* Класс сущность всей метаинформации по наборам данных  
 \* \*/  
public class MetaDataNoaa {  
 private Datasets formats;  
 private Datasets keywords;  
 private Datasets observationTypes;  
 private OffsetDateTime endDate;  
 private long count;  
 private Datasets datasets;  
 private List<Result> results;  
 private String startDate;  
  
 public MetaDataNoaa() {  
 }  
  
 @JsonProperty("formats")  
 public Datasets getFormats() { return formats; }  
 @JsonProperty("formats")  
 public void setFormats(Datasets value) { this.formats = value; }  
  
 @JsonProperty("keywords")  
 public Datasets getKeywords() { return keywords; }  
 @JsonProperty("keywords")  
 public void setKeywords(Datasets value) { this.keywords = value; }  
  
 @JsonProperty("observationTypes")  
 public Datasets getObservationTypes() { return observationTypes; }  
 @JsonProperty("observationTypes")  
 public void setObservationTypes(Datasets value) { this.observationTypes = value; }  
  
 @JsonProperty("endDate")  
 public OffsetDateTime getEndDate() { return endDate; }  
 @JsonProperty("endDate")  
 public void setEndDate(OffsetDateTime value) { this.endDate = value; }  
  
 @JsonProperty("count")  
 public long getCount() { return count; }  
 @JsonProperty("count")  
 public void setCount(long value) { this.count = value; }  
  
 @JsonProperty("datasets")  
 public Datasets getDatasets() { return datasets; }  
 @JsonProperty("datasets")  
 public void setDatasets(Datasets value) { this.datasets = value; }  
  
 @JsonProperty("results")  
 public List<Result> getResults() { return results; }  
 @JsonProperty("results")  
 public void setResults(List<Result> value) { this.results = value; }  
  
 @JsonProperty("startDate")  
 public String getStartDate() { return startDate; }  
 @JsonProperty("startDate")  
 public void setStartDate(String value) { this.startDate = value; }  
  
 @Override  
 public String toString() {  
 return "MetaDataNoaa{" +  
 "count=" + count +  
 ", startDate='" + startDate + '\'' +  
 '}';  
 }  
}

**Result.java**

package kg.npei.web.domain.Entities.NoaaEntities.MetaData;  
import com.fasterxml.jackson.annotation.\*;  
import java.time.LocalDate;  
import java.util.List;  
  
public class Result {  
 private boolean featured;  
 private List<Format> formats;  
 private List<Format> observationTypes;  
 private List<Format> keywords;  
 private LocalDate endDate;  
 private boolean available;  
 private String description;  
 private List<DataType> dataTypes;  
 private String name;  
 private List<String> parsedKeywords;  
 private Location location;  
 private Links links;  
 private String id;  
 private String doiLink;  
 private String startDate;  
 private String fileID;  
  
 @JsonProperty("featured")  
 public boolean getFeatured() { return featured; }  
 @JsonProperty("featured")  
 public void setFeatured(boolean value) { this.featured = value; }  
  
 @JsonProperty("formats")  
 public List<Format> getFormats() { return formats; }  
 @JsonProperty("formats")  
 public void setFormats(List<Format> value) { this.formats = value; }  
  
 @JsonProperty("observationTypes")  
 public List<Format> getObservationTypes() { return observationTypes; }  
 @JsonProperty("observationTypes")  
 public void setObservationTypes(List<Format> value) { this.observationTypes = value; }  
  
 @JsonProperty("keywords")  
 public List<Format> getKeywords() { return keywords; }  
 @JsonProperty("keywords")  
 public void setKeywords(List<Format> value) { this.keywords = value; }  
  
 @JsonProperty("endDate")  
 public LocalDate getEndDate() { return endDate; }  
 @JsonProperty("endDate")  
 public void setEndDate(LocalDate value) { this.endDate = value; }  
  
 @JsonProperty("available")  
 public boolean getAvailable() { return available; }  
 @JsonProperty("available")  
 public void setAvailable(boolean value) { this.available = value; }  
  
 @JsonProperty("description")  
 public String getDescription() { return description; }  
 @JsonProperty("description")  
 public void setDescription(String value) { this.description = value; }  
  
 @JsonProperty("dataTypes")  
 public List<DataType> getDataTypes() { return dataTypes; }  
 @JsonProperty("dataTypes")  
 public void setDataTypes(List<DataType> value) { this.dataTypes = value; }  
  
 @JsonProperty("name")  
 public String getName() { return name; }  
 @JsonProperty("name")  
 public void setName(String value) { this.name = value; }  
  
 @JsonProperty("parsedKeywords")  
 public List<String> getParsedKeywords() { return parsedKeywords; }  
 @JsonProperty("parsedKeywords")  
 public void setParsedKeywords(List<String> value) { this.parsedKeywords = value; }  
  
 @JsonProperty("location")  
 public Location getLocation() { return location; }  
 @JsonProperty("location")  
 public void setLocation(Location value) { this.location = value; }  
  
 @JsonProperty("links")  
 public Links getLinks() { return links; }  
 @JsonProperty("links")  
 public void setLinks(Links value) { this.links = value; }  
  
 @JsonProperty("id")  
 public String getID() { return id; }  
 @JsonProperty("id")  
 public void setID(String value) { this.id = value; }  
  
 @JsonProperty("doiLink")  
 public String getDoiLink() { return doiLink; }  
 @JsonProperty("doiLink")  
 public void setDoiLink(String value) { this.doiLink = value; }  
  
 @JsonProperty("startDate")  
 public String getStartDate() { return startDate; }  
 @JsonProperty("startDate")  
 public void setStartDate(String value) { this.startDate = value; }  
  
 @JsonProperty("fileId")  
 public String getFileID() { return fileID; }  
 @JsonProperty("fileId")  
 public void setFileID(String value) { this.fileID = value; }  
}

**Access.java**

package kg.npei.web.domain.Entities.NoaaEntities.MetaData;  
import com.fasterxml.jackson.annotation.\*;  
  
public class Access {  
 private String name;  
 private AccessType type;  
 private String url;  
  
 @JsonProperty("name")  
 public String getName() { return name; }  
 @JsonProperty("name")  
 public void setName(String value) { this.name = value; }  
  
 @JsonProperty("type")  
 public AccessType getType() { return type; }  
 @JsonProperty("type")  
 public void setType(AccessType value) { this.type = value; }  
  
 @JsonProperty("url")  
 public String getURL() { return url; }  
 @JsonProperty("url")  
 public void setURL(String value) { this.url = value; }  
}

**AccesType.java**

package kg.npei.web.domain.Entities.NoaaEntities.MetaData;  
import java.io.IOException;  
import com.fasterxml.jackson.annotation.\*;  
  
public enum AccessType {  
 DOWNLOAD, INFORMATION, ORDER, SEARCH;  
  
 @JsonValue  
 public String toValue() {  
 switch (this) {  
 case DOWNLOAD: return "download";  
 case INFORMATION: return "information";  
 case ORDER: return "order";  
 case SEARCH: return "search";  
 }  
 return null;  
 }  
  
 @JsonCreator  
 public static AccessType forValue(String value) throws IOException {  
 if (value.equals("download")) return DOWNLOAD;  
 if (value.equals("information")) return INFORMATION;  
 if (value.equals("order")) return ORDER;  
 if (value.equals("search")) return SEARCH;  
 throw new IOException("Cannot deserialize AccessType");  
 }  
}

**Bucket.java**

package kg.npei.web.domain.Entities.NoaaEntities.MetaData;  
import com.fasterxml.jackson.annotation.\*;  
  
public class Bucket {  
 private long docCount;  
 private String key;  
  
 @JsonProperty("docCount")  
 public long getDocCount() { return docCount; }  
 @JsonProperty("docCount")  
 public void setDocCount(long value) { this.docCount = value; }  
  
 @JsonProperty("key")  
 public String getKey() { return key; }  
 @JsonProperty("key")  
 public void setKey(String value) { this.key = value; }  
}

**Converter.java**

package kg.npei.web.domain.Entities.NoaaEntities.MetaData;  
import com.fasterxml.jackson.core.JsonParser;  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.\*;  
import com.fasterxml.jackson.databind.module.SimpleModule;  
  
import java.io.IOException;  
import java.time.OffsetDateTime;  
import java.time.OffsetTime;  
import java.time.ZoneOffset;  
import java.time.ZonedDateTime;  
import java.time.format.DateTimeFormatter;  
import java.time.format.DateTimeFormatterBuilder;  
import java.time.temporal.ChronoField;  
  
public class Converter {  
 // Date-time helpers  
  
 private static final DateTimeFormatter DATE\_TIME\_FORMATTER = new DateTimeFormatterBuilder()  
 .appendOptional(DateTimeFormatter.ISO\_DATE\_TIME)  
 .appendOptional(DateTimeFormatter.ISO\_OFFSET\_DATE\_TIME)  
 .appendOptional(DateTimeFormatter.ISO\_INSTANT)  
 .appendOptional(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss.SX"))  
 .appendOptional(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ssX"))  
 .appendOptional(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"))  
 .toFormatter()  
 .withZone(ZoneOffset.UTC);  
  
 public static OffsetDateTime parseDateTimeString(String str) {  
 return ZonedDateTime.from(Converter.DATE\_TIME\_FORMATTER.parse(str)).toOffsetDateTime();  
 }  
 private static final DateTimeFormatter TIME\_FORMATTER = new DateTimeFormatterBuilder()  
 .appendOptional(DateTimeFormatter.ISO\_TIME)  
 .appendOptional(DateTimeFormatter.ISO\_OFFSET\_TIME)  
 .parseDefaulting(ChronoField.YEAR, 2020)  
 .parseDefaulting(ChronoField.MONTH\_OF\_YEAR, 1)  
 .parseDefaulting(ChronoField.DAY\_OF\_MONTH, 1)  
 .toFormatter()  
 .withZone(ZoneOffset.UTC);  
  
 public static OffsetTime parseTimeString(String str) {  
 return ZonedDateTime.from(Converter.TIME\_FORMATTER.parse(str)).toOffsetDateTime().toOffsetTime();  
 }  
 // Serialize/deserialize helpers  
  
 public static MetaDataNoaa fromJsonString(String json) throws IOException {  
 return getObjectReader().readValue(json);  
 }  
  
 public static String toJsonString(MetaDataNoaa obj) throws JsonProcessingException {  
 return getObjectWriter().writeValueAsString(obj);  
 }  
  
 private static ObjectReader reader;  
 private static ObjectWriter writer;  
  
 private static void instantiateMapper() {  
 ObjectMapper mapper = new ObjectMapper();  
 mapper.findAndRegisterModules();  
 mapper.configure(SerializationFeature.WRITE\_DATES\_AS\_TIMESTAMPS, false);  
 SimpleModule module = new SimpleModule();  
 module.addDeserializer(OffsetDateTime.class, new JsonDeserializer<OffsetDateTime>() {  
 @Override  
 public OffsetDateTime deserialize(JsonParser jsonParser, DeserializationContext deserializationContext) throws IOException, JsonProcessingException {  
 String value = jsonParser.getText();  
 return Converter.parseDateTimeString(value);  
 }  
 });  
 mapper.registerModule(module);  
 reader = mapper.readerFor(MetaDataNoaa.class);  
 writer = mapper.writerFor(MetaDataNoaa.class);  
 }  
  
 private static ObjectReader getObjectReader() {  
 if (reader == null) instantiateMapper();  
 return reader;  
 }  
  
 private static ObjectWriter getObjectWriter() {  
 if (writer == null) instantiateMapper();  
 return writer;  
 }  
}

**Datasets.java**

package kg.npei.web.domain.Entities.NoaaEntities.MetaData;  
import com.fasterxml.jackson.annotation.\*;  
import java.util.List;  
  
public class Datasets {  
 private long docCountError;  
 private List<Bucket> buckets;  
 private long sumOfOtherDocCounts;  
  
 @JsonProperty("docCountError")  
 public long getDocCountError() { return docCountError; }  
 @JsonProperty("docCountError")  
 public void setDocCountError(long value) { this.docCountError = value; }  
  
 @JsonProperty("buckets")  
 public List<Bucket> getBuckets() { return buckets; }  
 @JsonProperty("buckets")  
 public void setBuckets(List<Bucket> value) { this.buckets = value; }  
  
 @JsonProperty("sumOfOtherDocCounts")  
 public long getSumOfOtherDocCounts() { return sumOfOtherDocCounts; }  
 @JsonProperty("sumOfOtherDocCounts")  
 public void setSumOfOtherDocCounts(long value) { this.sumOfOtherDocCounts = value; }  
}

**DataType.java**

package kg.npei.web.domain.Entities.NoaaEntities.MetaData;  
import com.fasterxml.jackson.annotation.\*;  
  
public class DataType {  
 private DateRange dateRange;  
 private long searchWeight;  
 private String name;  
 private String id;  
  
 @JsonProperty("dateRange")  
 public DateRange getDateRange() { return dateRange; }  
 @JsonProperty("dateRange")  
 public void setDateRange(DateRange value) { this.dateRange = value; }  
  
 @JsonProperty("searchWeight")  
 public long getSearchWeight() { return searchWeight; }  
 @JsonProperty("searchWeight")  
 public void setSearchWeight(long value) { this.searchWeight = value; }  
  
 @JsonProperty("name")  
 public String getName() { return name; }  
 @JsonProperty("name")  
 public void setName(String value) { this.name = value; }  
  
 @JsonProperty("id")  
 public String getID() { return id; }  
 @JsonProperty("id")  
 public void setID(String value) { this.id = value; }  
}

**DateRange.java**

package kg.npei.web.domain.Entities.NoaaEntities.MetaData;  
import com.fasterxml.jackson.annotation.\*;  
  
public class DateRange {  
 private Object start;  
 private Object end;  
  
 @JsonProperty("start")  
 public Object getStart() { return start; }  
 @JsonProperty("start")  
 public void setStart(Object value) { this.start = value; }  
  
 @JsonProperty("end")  
 public Object getEnd() { return end; }  
 @JsonProperty("end")  
 public void setEnd(Object value) { this.end = value; }  
}

**Format.java**

package kg.npei.web.domain.Entities.NoaaEntities.MetaData;  
import com.fasterxml.jackson.annotation.\*;  
  
public class Format {  
 private String name;  
 private String id;  
  
 @JsonProperty("name")  
 public String getName() { return name; }  
 @JsonProperty("name")  
 public void setName(String value) { this.name = value; }  
  
 @JsonProperty("id")  
 public String getID() { return id; }  
 @JsonProperty("id")  
 public void setID(String value) { this.id = value; }  
}

**Links.java**

package kg.npei.web.domain.Entities.NoaaEntities.MetaData;  
import com.fasterxml.jackson.annotation.\*;  
import kg.npei.web.domain.Entities.NoaaEntities.MetaData.Access;  
import java.util.List;  
  
public class Links {  
 private List<Access> other;  
 private List<Access> access;  
 private List<Access> documentation;  
  
 @JsonProperty("other")  
 public List<Access> getOther() { return other; }  
 @JsonProperty("other")  
 public void setOther(List<Access> value) { this.other = value; }  
  
 @JsonProperty("access")  
 public List<Access> getAccess() { return access; }  
 @JsonProperty("access")  
 public void setAccess(List<Access> value) { this.access = value; }  
  
 @JsonProperty("documentation")  
 public List<Access> getDocumentation() { return documentation; }  
 @JsonProperty("documentation")  
 public void setDocumentation(List<Access> value) { this.documentation = value; }  
}

**Location.java**

package kg.npei.web.domain.Entities.NoaaEntities.MetaData;  
import com.fasterxml.jackson.annotation.\*;  
import java.util.List;  
  
public class Location {  
 private List<List<List<Double>>> coordinates;  
 private LocationType type;  
  
 @JsonProperty("coordinates")  
 public List<List<List<Double>>> getCoordinates() { return coordinates; }  
 @JsonProperty("coordinates")  
 public void setCoordinates(List<List<List<Double>>> value) { this.coordinates = value; }  
  
 @JsonProperty("type")  
 public LocationType getType() { return type; }  
 @JsonProperty("type")  
 public void setType(LocationType value) { this.type = value; }  
}

**LocationType.java**

package kg.npei.web.domain.Entities.NoaaEntities.MetaData;  
import java.io.IOException;  
import com.fasterxml.jackson.annotation.\*;  
  
public enum LocationType {  
 POLYGON;  
  
 @JsonValue  
 public String toValue() {  
 switch (this) {  
 case POLYGON: return "polygon";  
 }  
 return null;  
 }  
 @JsonCreator  
 public static LocationType forValue(String value) throws IOException {  
 if (value.equals("polygon")) return POLYGON;  
 throw new IOException("Cannot deserialize LocationType");  
 }  
}

Интерфейсы

**ServiceInitDataSetInterfaceStations.java**

package kg.npei.web.services;  
import org.springframework.ui.Model;  
import java.io.IOException;  
  
public interface ServiceInitDataSetInterfaceStations {  
 void addParam(Model model) throws IOException;  
}

**ServiceInitInterface.java**

package kg.npei.web.services;  
import kg.npei.web.domain.Entities.NoaaEntities.KeywordsDatasets;  
import org.springframework.ui.Model;  
import java.io.IOException;  
  
public interface ServiceInitInterface {  
 void addParam(Model model) throws IOException;  
 KeywordsDatasets getKey();  
}

**ServiceInitInterfaceDataSets.java**

package kg.npei.web.services;  
import org.springframework.ui.Model;  
import java.io.IOException;  
  
public interface ServiceInitInterfaceDataSets {  
 void addParam(Model model) throws IOException;  
}

**ServiceInitInterfaceMetaData.java**

package kg.npei.web.services;  
import org.springframework.ui.Model;  
import java.io.IOException;  
  
public interface ServiceInitInterfaceMetaData {  
 void addParam(Model model) throws IOException;  
}

**ServiceInitMetaDataSets.java**

package kg.npei.web.services;  
import org.springframework.ui.Model;  
import java.io.IOException;  
  
public interface ServiceInitMetaDataSets {  
 void addParam(Model model) throws IOException;  
}

**ServiceInitStaticDataSets.java**

package kg.npei.web.services;  
import kg.npei.web.domain.Entities.NoaaEntities.DataSetAndDataTypes.DataSetAndDataTypes;  
import kg.npei.web.domain.Entities.NoaaEntities.MetaData.MetaDataNoaa;  
import kg.npei.web.parser.JacksonParseFromFileDataSetsAndTypes;  
import kg.npei.web.parser.JacksonParseFromFileMetaData;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.ui.Model;  
import java.io.IOException;  
  
@Service  
public class ServiceInitStaticDataSets implements ServiceInitInterfaceDataSets {  
 @Autowired  
 private ServiceInitInterfaceDataSets serviceInitInterfaceDataSets;  
 private DataSetAndDataTypes dataSetAndDataTypes;  
  
 @Override  
 public void addParam(Model model) throws IOException {  
 dataSetAndDataTypes = null;  
 JacksonParseFromFileDataSetsAndTypes jacksonParseFromFileDataSetsAndTypes = new JacksonParseFromFileDataSetsAndTypes();  
 jacksonParseFromFileDataSetsAndTypes.parseJsonFromFile("D:\\allAplication\\npeiKG\\src\\main\\resources\\resourcesDataSetAndDataTypes.json");  
 dataSetAndDataTypes = jacksonParseFromFileDataSetsAndTypes.getDataSetAndDataTypes();  
 model.addAttribute("dataSetAndDataTypes",dataSetAndDataTypes);  
 }  
}

**ServiceInitStaticDataSetStations.java**

package kg.npei.web.services;  
import kg.npei.web.domain.Entities.NoaaEntities.DataSetAndDataTypes.DataSetAndDataTypes;  
import kg.npei.web.domain.Entities.NoaaEntities.Station.StationZC;  
import kg.npei.web.parser.JacksonParseFromFileDataSetsAndTypes;  
import kg.npei.web.parser.JacksonParseFromFileStationZC;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.ui.Model;  
import java.io.IOException;  
  
@Service  
public class ServiceInitStaticDataSetStations implements ServiceInitDataSetInterfaceStations {  
 @Autowired  
 private ServiceInitDataSetInterfaceStations serviceInitDataSetInterfaceStations;  
  
 private StationZC stationZC;  
  
 @Override  
 public void addParam(Model model) throws IOException {  
 stationZC = null;  
 JacksonParseFromFileStationZC jacksonParseFromFileStationZC = new JacksonParseFromFileStationZC();  
 jacksonParseFromFileStationZC.parseJsonFromFile("D:\\allAplication\\npeiKG\\src\\main\\resources\\resourcesStationZC.json");  
 stationZC = jacksonParseFromFileStationZC.getStationZC();  
 model.addAttribute("stationZC",stationZC);  
 }  
}

**JacksonParser.java**

package kg.npei.web.parser;  
import kg.npei.web.domain.Entities.NoaaEntities.MetaDataAbstract;  
  
import java.io.IOException;  
  
public interface JacksonParser<T> {  
 public void ParseJson(T metaData, String url, String nameFile)throws IOException;  
 public void ParseJson(T metadata,String url) throws IOException;  
 T getMeta();  
}

**JacksonParseMetaDataNoaa.java**

package kg.npei.web.parser;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;  
import kg.npei.web.domain.Entities.NoaaEntities.MetaData.MetaDataNoaa;  
import kg.npei.web.domain.Entities.NoaaEntities.MetaDataAbstract;  
  
import java.io.DataInputStream;  
import java.io.File;  
import java.io.IOException;  
import java.net.URL;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.nio.file.StandardOpenOption;  
/\*\*  
 \* Парсинг json файла  
 \* \*/  
public class JacksonParseMetaDataNoaa<T> implements JacksonParser<T>{  
 @Override  
 public void ParseJson(T metaDataNoaa, String url, String nameFile)throws IOException {  
 ObjectMapper objectMapper = new ObjectMapper();  
 objectMapper.registerModule(new JavaTimeModule());  
 metaDataNoaa = (T) objectMapper.readValue(new URL(url),MetaDataNoaa.class);  
  
 String jsonString = objectMapper.writerWithDefaultPrettyPrinter()  
 .writeValueAsString(metaDataNoaa);  
 DataInputStream dis = new DataInputStream(null);  
  
 Files.write(Paths.get(new File(new File("./src/main/resources/resources/").getAbsolutePath()).getPath()+nameFile), jsonString.getBytes() , StandardOpenOption.CREATE);  
 }  
 private T meta=null;  
  
 @Override  
 public T getMeta() {  
 return meta;  
 }  
  
 @Override  
 public void ParseJson(T metaDataNoaa, String url)throws IOException {  
 ObjectMapper objectMapper = new ObjectMapper();  
 objectMapper.registerModule(new JavaTimeModule());  
 metaDataNoaa = (T) objectMapper.readValue(new URL(url),metaDataNoaa.getClass());  
 meta = metaDataNoaa;  
 }  
}

**JacksonParseMetaDataSets.java**

package kg.npei.web.parser;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import kg.npei.web.domain.Entities.NoaaEntities.MetaDataSets.Converter;  
import kg.npei.web.domain.Entities.NoaaEntities.MetaDataSets.MetaDataSets;  
  
import java.io.File;  
import java.io.IOException;  
import java.net.URL;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.nio.file.StandardOpenOption;  
  
import com.fasterxml.jackson.databind.\*;  
import com.fasterxml.jackson.databind.module.SimpleModule;  
import com.fasterxml.jackson.core.JsonParser;  
import com.fasterxml.jackson.core.JsonProcessingException;  
import org.springframework.beans.factory.annotation.Autowired;  
  
import java.time.OffsetDateTime;  
  
public class JacksonParseMetaDataSets<T> implements JacksonParser<T>{  
 @Autowired  
 private JacksonParser jacksonParser;  
 private MetaDataSets meta;  
  
 @Override  
 public void ParseJson(T metaDataSets, String url, String nameFile)throws IOException {  
 ObjectMapper objectMapper = new ObjectMapper();  
 objectMapper.findAndRegisterModules();  
 objectMapper.configure(SerializationFeature.WRITE\_DATES\_AS\_TIMESTAMPS, false);  
 SimpleModule module = new SimpleModule();  
 module.addDeserializer(OffsetDateTime.class, new JsonDeserializer<OffsetDateTime>() {  
 @Override  
 public OffsetDateTime deserialize(JsonParser jsonParser, DeserializationContext deserializationContext) throws IOException, JsonProcessingException {  
 String value = jsonParser.getText();  
 return Converter.parseDateTimeString(value);  
 }  
 });  
 objectMapper.registerModule(module);  
  
 metaDataSets = (T) objectMapper.readValue(new URL(url), MetaDataSets.class);  
 meta = new MetaDataSets();  
 String jsonString = objectMapper.writerWithDefaultPrettyPrinter()  
 .writeValueAsString(metaDataSets);  
 meta = new MetaDataSets();  
 meta = (MetaDataSets) metaDataSets;  
 MetaDataSets data = Converter.fromJsonString(jsonString);  
 metaDataSets = (T)data;  
  
 }  
  
 @Override  
 public void ParseJson(T metaDataSets, String url)throws IOException {  
 }  
  
 @Override  
 public T getMeta() {  
 return (T)meta;  
 }  
}

**ServiceNoaaDataSetsImpl.java**

package kg.npei.web.services;  
import kg.npei.web.domain.Entities.NoaaEntities.MetaData.MetaDataNoaa;  
import kg.npei.web.domain.Entities.NoaaEntities.MetaDataAbstract;  
import kg.npei.web.domain.Entities.RequestSearchParam;  
import kg.npei.web.parser.JacksonParseMetaDataNoaa;  
import kg.npei.web.parser.JacksonParser;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.ui.Model;  
import org.springframework.web.servlet.ModelAndView;  
  
import javax.servlet.http.HttpServletRequest;  
import java.io.IOException;  
  
@Service  
public class ServiceNoaaDataSetsImpl implements ServiceNoaaParametersInterface {  
  
 @Autowired  
 private ServiceNoaaParametersInterface serviceNoaaParametersInterface;  
 private RequestSearchParam requestSearchParam;  
  
 private MetaDataAbstract mdA;  
 private MetaDataNoaa NoaaDataSets;  
  
  
 @Override  
 public void addParametrs(ModelAndView modelAndView) {}  
  
 @Override  
 public void addParametrs(HttpServletRequest httpServletRequest, Model model){  
  
 NoaaDataSets = new MetaDataNoaa();  
 requestSearchParam = new RequestSearchParam();  
 System.out.println(httpServletRequest.getParameter("startDate"));  
 System.out.println(httpServletRequest.getParameter("startDate"));  
//  
// requestSearchParam.setStartDate(httpServletRequest.getParameter("startDate"));  
// requestSearchParam.setEndDate(httpServletRequest.getParameter("endDate"));  
 requestSearchParam.setBoundingBox(httpServletRequest.getParameter("keywords"));  
// (N,W,S,E)  
 requestSearchParam.setBoundingBox(httpServletRequest.getParameter("N") + ","+httpServletRequest.getParameter("W") + "," +  
 httpServletRequest.getParameter("S") +","+ httpServletRequest.getParameter("E"));  
 requestSearchParam.setLimit("1000");  
  
  
 model.addAttribute("requestParam",requestSearchParam);  
  
 requestSearchParam.setBaseUrl("https://www.ncei.noaa.gov/access/services/search/v1/");  
 System.out.println(requestSearchParam.getHttpLink());  
 JacksonParser jacksonParser = new JacksonParseMetaDataNoaa();  
 // mdA = new MetaDataNoaa();  
 try {  
 jacksonParser.ParseJson(NoaaDataSets,requestSearchParam.getHttpLink());  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 model.addAttribute("NoaaDataSets",NoaaDataSets);  
 System.out.println(requestSearchParam.getHttpLink());  
 }  
  
 @Override  
 public RequestSearchParam getRequestSearchParam(){  
 return requestSearchParam;  
 }  
}