## СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 4](#_Toc530076940)

[ВВЕДЕНИЕ 6](#_Toc530076941)

[1.АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ 7](#_Toc530076942)

[1.1. Визуализация данных 7](#_Toc530076943)

[1.2. Анализ существующих аналогов 10](#_Toc530076944)

[1.3. Постановка задачи 13](#_Toc530076945)

[2. МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ 14](#_Toc530076946)

[2.1. Представление многомерной информации на карте 14](#_Toc530076947)

[2.2. Описание функциональности ПС 16](#_Toc530076948)

[2.3. Спецификация функциональных требований 18](#_Toc530076949)

[3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 19](#_Toc530076950)

[3.1. Проектирование классов проекта 19](#_Toc530076951)

[3.2. Разработка взаимодействия с внешними API 20](#_Toc530076952)

[3.2.1. Взаимодействие с API карт 20](#_Toc530076956)

[3.2.2. Взаимодействие с API Nominatim 21](#_Toc530076961)

[3.3. Разработка алгоритма генерации базовых цветов 21](#_Toc530076967)

[3.4. Разработка алгоритма осветления цветов 22](#_Toc530076972)

[3.5. Разработка алгоритма смешения цветов 23](#_Toc530076975)

[3.6. Разработка алгоритма получения списка регионов 24](#_Toc530076976)

[3.7. Разработка алгоритма подсчета коэффициента осветления цвета 25](#_Toc530076977)

[3.7.1. Разработка алгоритма подсчета коэффициента осветления на основе общего числа повторений 25](#_Toc530076978)

[3.7.2. Разработка алгоритма подсчета коэффициента осветления на основе количества повторений ситуаций на душу населения. 26](#_Toc530076979)

[4. КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 27](#_Toc530076980)

[4.1. Обоснование выбора средств разработки ПС 27](#_Toc530076981)

[4.2. Описание классов и методов 31](#_Toc530076982)

[4.2.1. Описание класса Region 31](#_Toc530076983)

[4.2.2. Описание класса Color 31](#_Toc530076984)

[4.2.3. Описание класса RegionsList 32](#_Toc530076985)

[4.2.4. Описание интерфейса CoefficientsСalculator 33](#_Toc530076986)

[4.2.5. Описание интерфейса MaxArrCalculator 34](#_Toc530076992)

[4.2.6. Описание класса HttpInteraction 34](#_Toc530076996)

[4.2.7. Описание интерфейса APICommunication 34](#_Toc530076997)

[4.2.8. Описание класса MainController 35](#_Toc530076998)

[4.2.9. Описание класса DBApiController 35](#_Toc530076999)

[5. ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ 37](#_Toc530077000)

[5.1. Тестирование сервера с помощью модульного тестирования 37](#_Toc530077001)

[5.2. Тестирование целостной системы 37](#_Toc530077002)

[6. РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ 40](#_Toc530077003)

[6.1. Серверная часть 40](#_Toc530077004)

[6.2. Клиентская часть 41](#_Toc530077005)

[6.3. API 43](#_Toc530077006)

[ЗАКЛЮЧЕНИЕ 45](#_Toc530077007)

[СПИСОК ЛИТЕРАТУРЫ 46](#_Toc530077008)

[ПРИЛОЖЕНИЕ 1 47](#_Toc530077009)

[ПРИЛОЖЕНИЕ 2 48](#_Toc530077010)

# ВВЕДЕНИЕ

Каждый день человек потребляет огромное количество информации: от этикеток на продуктах до отчетов Всемирной организации здравоохранения. В результате, предоставить информацию так, чтобы ее можно было легко воспринимать, становится все сложнее и сложнее.

Один из лучших способов помочь воспринять информацию — визуализация этих данных. Кроме того, наглядная демонстрация помогает заметить различные закономерности, которые было бы сложно увидеть из обычных статистических данных.

Визуализация — важнейший этап в процессе анализа данных. Он помогает представить результаты исследований в простой и понятной форме, зачастую служит ключевым фактором для принятия решений в различных сферах деятельности.

Данная курсовая работа посвящена разработке веб-сервиса для представления многомерной информации на карте.

В ходе выполнения данного проекта я постараюсь понять:

1. Как разработать веб-сервис на Java Spring
2. Как эффективно обработать большие объемы информации
3. Как работать с API карт
4. Как спроектировать свой REST API сервис

В этой пояснительной записке отображены следующие этапы написания курсовой работы:

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;
2. Анализ требований к программному средству и разработка функциональных требований;
3. Проектирование программного средства;
4. Создание (конструирование) программного средства;
5. Тестирование, проверка работоспособности и анализ полученных результатов;
6. Руководство по установке и использованию.

# АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

## Визуализация данных

Визуализация любой задачи требует индивидуального подхода в зависимости от поставленной задачи и важно на ранней стадии внедрения визуализации определить оптимальный способ.

Существуют разные классификации способов визуализации данных.

Наиболее интересны две общие категории: исследование(exploration) и толкование(explanation)[1] Они служат концептуально различным задачам визуализации, что сказывается на всех характеристиках метода: от инструментов до дизайна.

“Исследование” - метод, который полезен для изучения неизвестных наборов данных: перенос в визуальное пространство помогает идентифицировать данные. В том числе их функции, тенденции и аномальные выбросы. Этот метод чаще всего является частью фазы анализа данных и используется для определения истории, которую данные могут рассказать. Пример визуализации данных, основанном на данном методе представлен на рисунке 1.1.

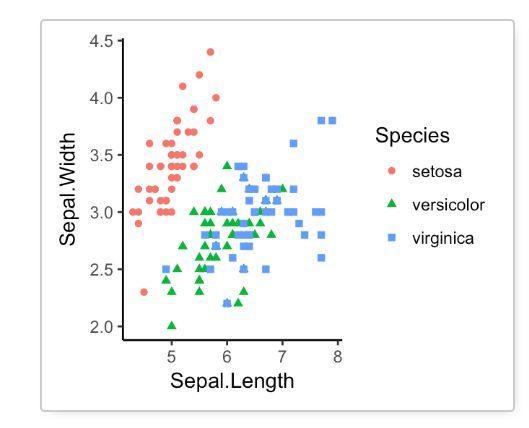
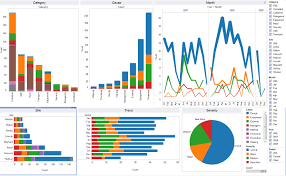
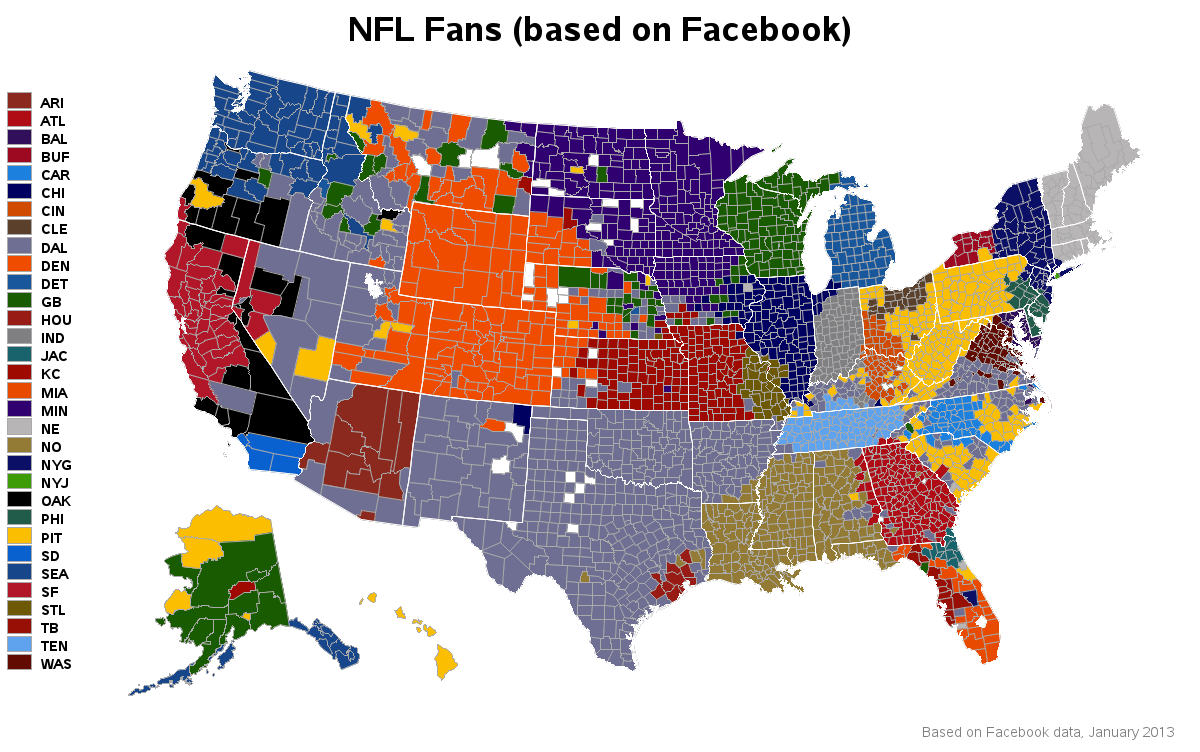


Рисунок 1.1 – Пример визуализации данных, основанном на методе «Exploration»

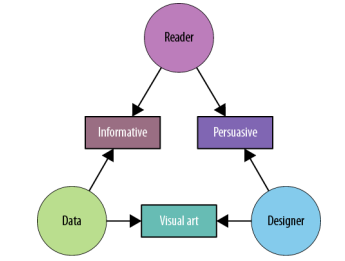
“Толкование” - метод, который используется не для изучения данных, а для передачи информации об этих данных. Особенностью метода является то, что проектировщику заранее известны все свойства данных и он может максимально эффективно представить данные. Пример визуализации данных, основанном на данном методе представлен на рисунке 1.2.

Рисунок 1.2 – Пример визуализации данных, основанном на методе «Explanation»

Существует также гибридная категория. Такие визуализации обычно интерактивны через графический интерфейс, который позволяет пользователю выбирать и ограничивать определенные параметры, тем самым обнаруживая для себя любые идеи, которые может предложить набор данных. В этих гибридах существует определенная свобода для исследования информации, но обычно она не является полностью сырой, а в какой-то степени перегоняется и облегчается. Пример гибридной визуализации приведен на рисунке 1.3.

Рисунок 1.3 – Пример гибридной визуализации данных

Эффективная визуализация данных состоит из трех компонентов: дизайнера, читателя и данных [1]. Схема представлена на рисунке 1.4. Каждый из «компонентов» вносят свой «вклад» в то, чтобы конечный пользователь получил наиболее полезный результат. Каждый из трех «компонентов» имеет свое уникальное отношение с двумя другими и необходимо учитывать потребности и перспективы всех трех при каждой визуализации. «Доминирующее» отношение определяет, какая будет категория визуализации.

Рисунок 1.4. - Схема связи дизайнера, читателя и данных

«Познавательная визуализация» (Informative visualization) в первую очередь служит для взаимосвязи читателя и данных. Она направлена на нейтральные представления фактов таким образом, чтобы обучить читателя.

«Убедительная визуализация» (Persuasive visualization) в первую очередь служит для взаимосвязи дизайнера и читателя. Это полезно, когда дизайнер хочет изменить мнение читателя о чем-либо. Визуализация представляет собой очень конкретную точку зрения и выступает за изменение мнения или за побуждение действия со стороны читателя.

«Визуальное искусство» (Visual Art) отличается от предыдущих двух категорий тем, что оно часто влечет за собой однонаправленное кодирование информации, что означает, что читатель не сможет декодировать визуальное представление так, чтобы понять основную информацию, в то время как информативные и убедительные визуализации должны быть легко двунаправленно декодированными.

## Анализ существующих аналогов

После тщательных поисков мною не было найдено сервисов для визуализации многомерной информации на карте, поэтому я рассмотрю просто сервисы для представления данных на карте.

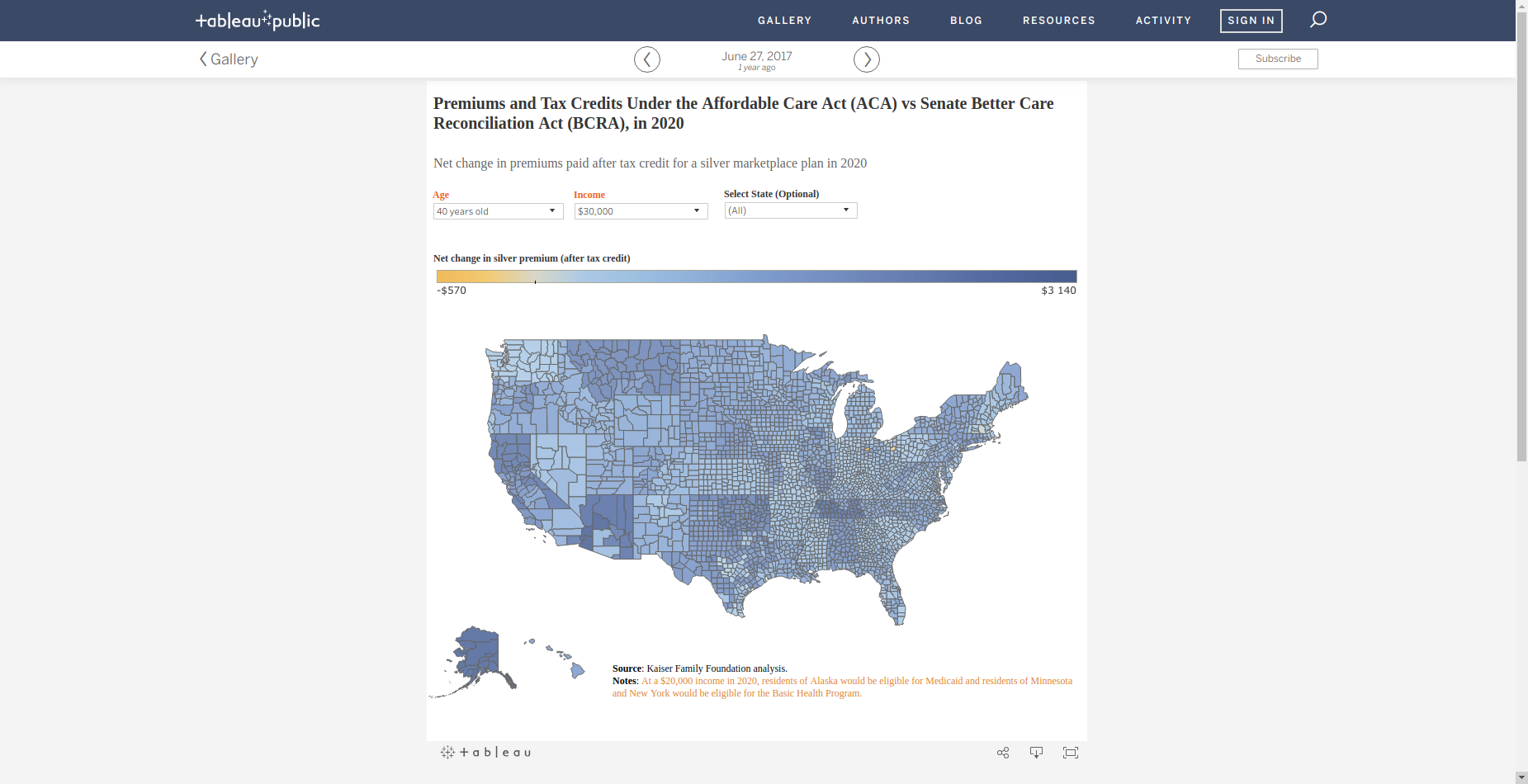
 **Tableu Public** — это популярный инструмент для визуализации данных, и, к тому же, практически бесплатный. Пользователи легко могут загрузить информацию в систему и наблюдать, как они обновляются в режиме реального времени. А для ускорения процесса можно скооперироваться с другими участникам проекта.

Рисунок 1.5 – Скриншот визуализации с помощью сервиса Tableu

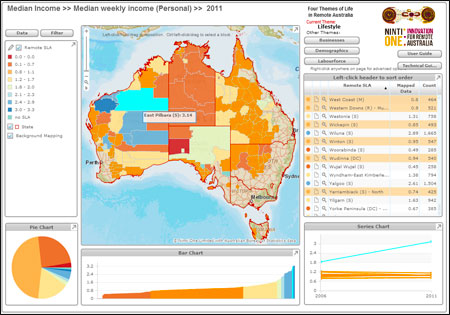
Плюсы:

* Большая часть функционала доступна бесплатно
* Скорость работы
* Обработка данных в режиме реального времени
* Масштабируемость

Минусы:

* Одномерность визуализируемой информации
* Необходимо устанавливать программу для визуализации

**InstantAtlas** — инструмент, который делает возможным создание высоко-интерактивных динамических и профильных отчетов, объединяющих статистику и картографическую информацию для качественной визуализации данных.

Рисунок 1.5 – Скриншот визуализации с помощью сервиса InstantAtlas

Плюсы:

* Скорость работы
* Широкий спектр функций
* Масштабируемость

Минусы:

* Одномерность визуализируемой информации
* Необходимо устанавливать программу для визуализации

**Google Chart Tools API** – это многофункциональный набор инструментов для визуализации данных. С помощью него можно относительно легко строить графики и диаграммы на сайте.

Функционал Google Chart Tools API включает в себя:

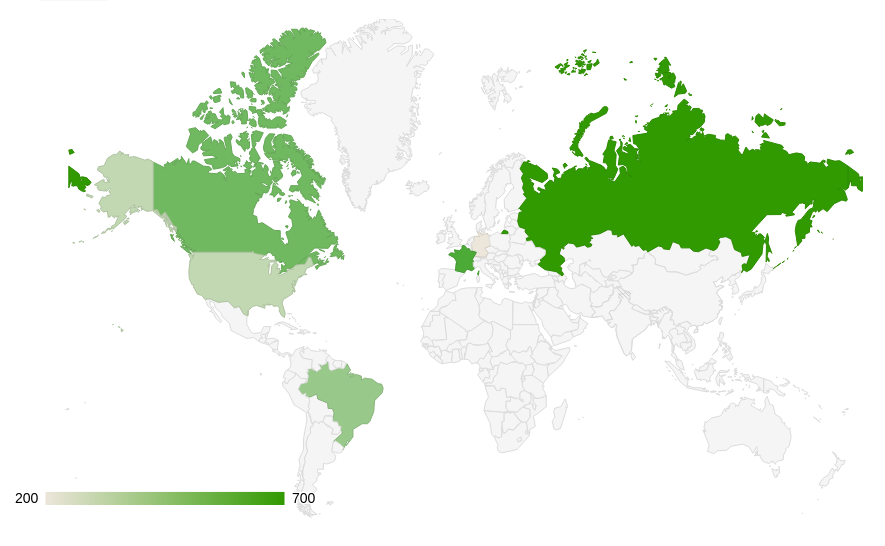
* Динамические пиктограммы;
* Карты;
* Циферблаты и дисплеи;
* Возможность создавать свои инструменты визуализации и использовать сторонние.

Рисунок 1.6 – Скриншот визуализации с помощью сервиса Google Charts

Плюсы:

* Бесплатный
* Большое количество функций
* Удобный API
* Скорость работы

Минусы:

* Одномерность визуализируемой информации

## Постановка задачи

Так как проектируемое программное средство должно визуализировать многомерную информацию, необходимо сделать удобную расшифровку данной информации. Программа должна расшифровывать многомерную информацию по региону при нажатии на него.

В данном сервисе должны быть реализованы следующие функции:

* Отображение данных на карте
* Обработка исходных данных в соответствие с запросом пользователя
* Отсутствие привязки к конкретной задачи (Массовость сервиса)
* Отсутствие привязки к конкретной карте
* Разработка удобного API
* Изменение запросов пользователя без перезагрузки страницы

Должны присутствовать следующие методы API:

* Получение обработанных данных за заданный период
* Получение «сырых» данных статистики за заданный период
* Получение «сырых» данных статистики конкретного региона
* Получение списка «ситуаций» (под ситуацией понимается конкретная единица многомерных данных)
* Получение количества ситуаций
* Получение полигона региона
* Получение числа жителей региона

Так же должна присутствовать документация к методам API сервиса, включающая в себе тип запроса, принимаемые параметры и описание.

Код должен быть сопровожден юнит-тестами и быть читабельным и простым в модифицировании.

В качестве основного языка программирования выбран язык Java, так как это удобный, довольно быстрый и функциональный язык со строгой статической типизацией. Дополнительно будут использоваться такие языки, как JS, SQL.

# МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

## Представление многомерной информации на карте

В настоящее время объем необходимых для обработки данных стремительно растут, однако общеупотребительные методы их представления и обработки требуют много ресурсов и времени, а также недостаточно просты для понимания [2]

Одним из решений этой проблемы может стать представление данных в графическом виде на пространственно-временной поверхности.

Рассмотрим использование данного метода на примере представлении комплексной статистики по пожарам на карте.

Входными данными будем считать:

* База данных МЧС РБ по пожарам за последние 11 лет;
* GET-параметры с выбранными пользователем фильтрами;

В пространственной плоскости S данные представляются согласно административно-территориальному делению.

Разобьем плоскость на m частей, где Si - представление на карте частотной и количественной информации за период.

Каждый Si является функцией. Si = f(x, y, z, c), где x, y, z - географические координаты, c - полученный цвет, является функцией C (Si1, Si2, …, Sin)

В свою очередь Si-ую часть можно разбить на N частей, где N - количество ситуаций.

Каждый Sij-ый элемент хранит количество повторений j-ой ситуации в i-ом регионе. Найдем максимальное число повторений каждой ситуации по формуле (2.1).

Mj = Max(Mij) (2.1)

Каждой ситуации сопоставим насыщенный цвет λj - данный цвет соответствует 100%-ому объему j-ой ситуации.

Для каждой j-ой ситуации i-ого региона найдем его цвет χij путем осветления базового цвета λj на η%, где

η = 100 - (Sij / Mj) \* 100

После чего найдем цвет каждого i-ого региона путем смешения χij-ых цветов методом суммы их RGB-представления.

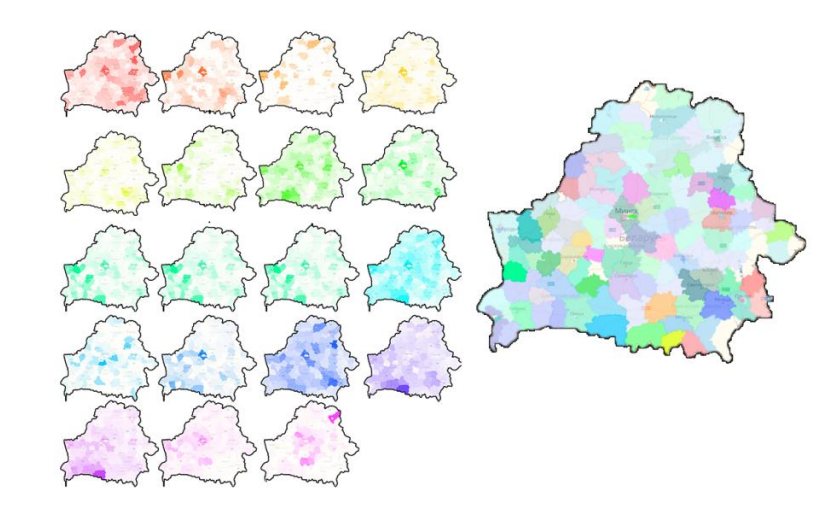
, (2.2)

, (2.3)

**,** (2.4)

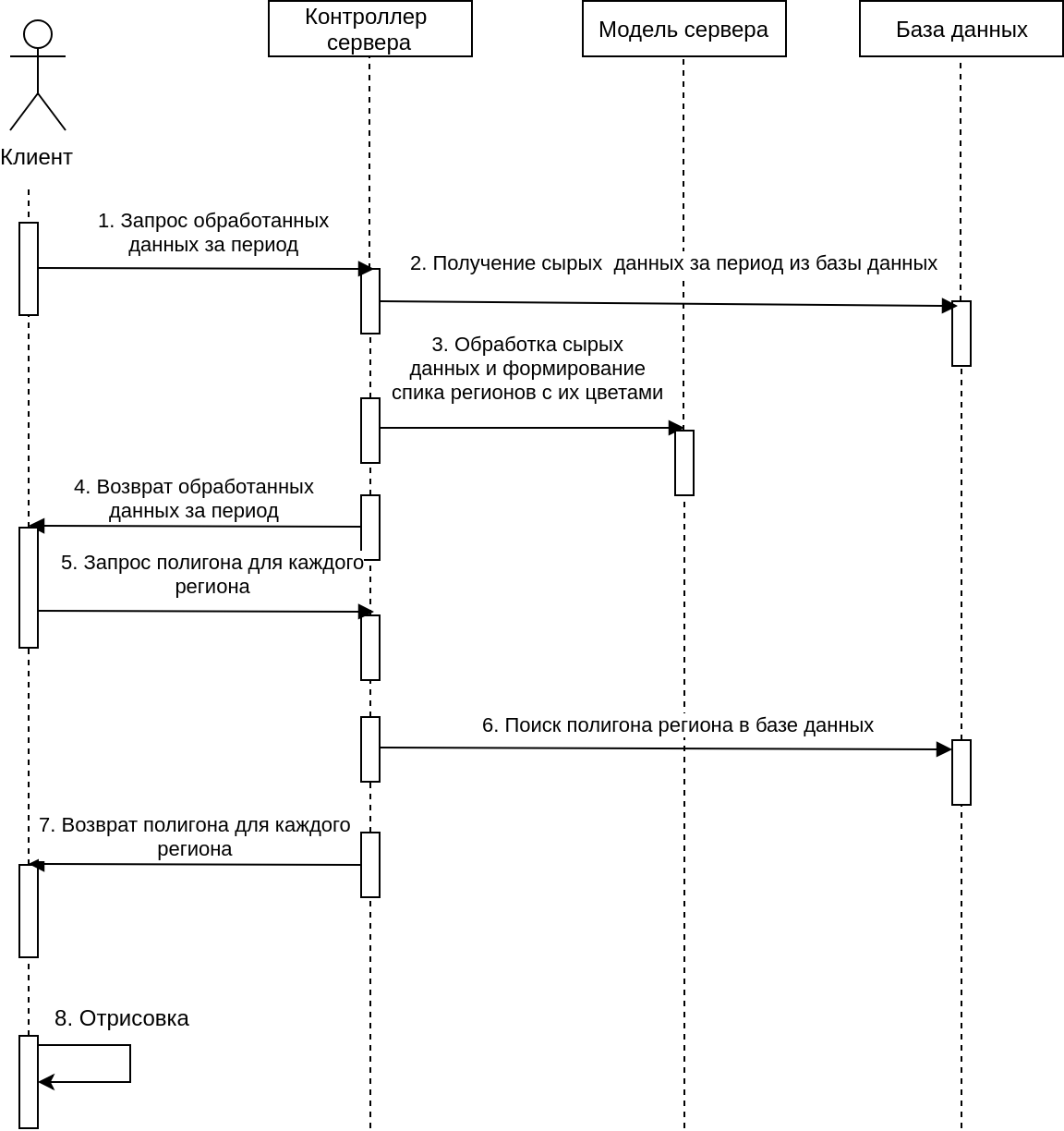
Пример такого смешения цветов представлен на рисунке 2.1.

После получения всех входных параметров функции Si, она наносится на пространственно-временную поверхность.

Рисунок 2.1. Пример смешения цветов

## Описание функциональности ПС

Основная задача сервиса — визуализировать заданную в базе данных информацию. Веб сервис должен предоставлять возможность выбора диапазона дат для отображения выборки из статистики. Выбор диапазона должен производиться в интерактивном текстовом поле. После выбора диапазона, должен производиться запрос на сервер, который вернет сформированные данные, после чего клиенту остается только получить полигон для каждого региона и нанести каждый конкретный регион на карту. Диаграмма последовательности приведена на рисунке 2.2.

Рисунок 2.2. Диаграмма последовательности процесса визуализации

Для функционирования API, необходимы следующие методы: получение сырых данных из базы данных (на основе диапазона дат, региона), получение сформированных данных, получение списка и числа ситуаций, получение полигона для региона, получение количества жителей региона. Use Case диаграмма приведена на рисунке 2.3.

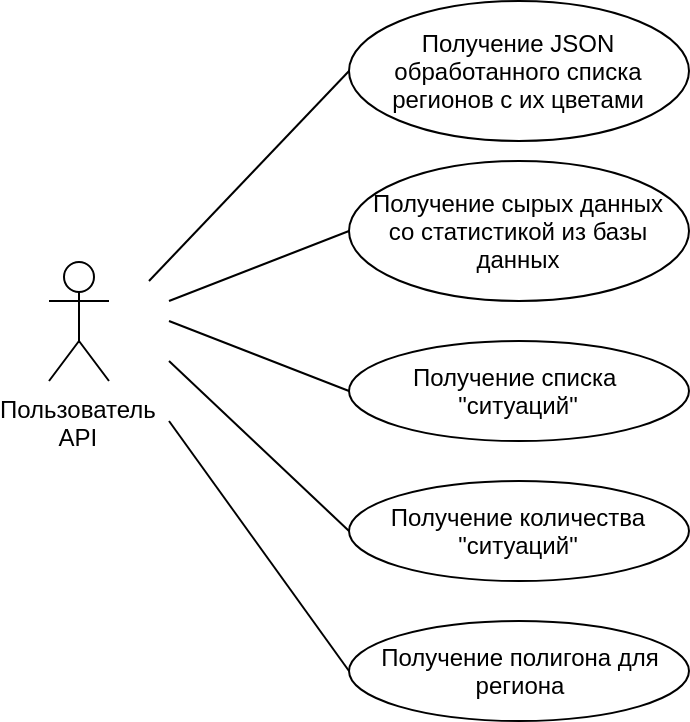


Рисунок 2.3. Use Case диаграмма требуемых возможностей API

Среди таблиц базы данных должны присутствовать следующие:

* Таблица с «черновыми» данными статистики
  + Поле «id»
  + Поле «Название региона»
  + Поле «ID ситуации»
  + Поле «Дата»
* Таблица «Ситуаций»
  + Поле «id» (Соответствует «ID ситуаций» в пред. Таблице)
  + Поле «Название»
* Таблица полигонов
  + Поле «id»
  + Поле «Название региона»
  + Поле «Полигон»
* Таблица популяции регионов
  + Поле «id»
  + Поле «Название региона»
  + Поле «Популяция»

## Спецификация функциональных требований

Среди функциональных требований есть «Визуализация заданной в БД информации».

Спецификация данной функции может иметь следующий вид:

* Диапазон дат, за который смотрится статистика, выбирает пользователь.
* Изначально, каждой «ситуации» соответствует свой уникальный цвет
* Каждый регион, за который в базе данных есть статистика за заданный период, должен быть раскрашен своим цветом
* Цвет региона получается путем смешения цветов «ситуаций» региона (который в свою очередь получается путем осветления базового цвета «ситуации» на основании интенсивности повторений «ситуаций» в регионе за заданный период)
* Пользователь задает принцип определения интенсивности повторений (либо общее количество повторений, либо на основе демографии в регионе)
* Пользователь должен иметь возможность посмотреть детализацию по региону путем клика по нужному региону
* Детализация должна представлять собой расшифровку цвета региона: количество повторений той или иной «ситуации»

При необходимости, пользователь должен иметь возможность изменять диапазон «на лету» без перезагрузки страницы

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## Проектирование классов проекта

На основе функциональных требований, необходимо спроектировать основные классы модели программного продукта.

В программном продукте предполагаются следующие основные классы модели:

* Класс «Color»
  + Класс для работы с цветами: преобразования между разными представлениями цветов, осветление, смешение цветов
* Класс «Region»
  + Класс региона, в котором хранится основная информация о регионе: название, цвет.
* Класс «RegionList»
  + В данном классе формируется список регионов и просчитывается цвет для каждого регионов. Объект данного класса является «выходным» по запросу обработанных данных из API или по запросу визуализации данных
* Интерфейс CoefficientCalculators
  + Используется для подсчёта коэффициента осветления цвета в соответствие с некоторыми параметрами (количеством повторений/количеством повторений учитывая демографию региона (количество повторений на число человек))
  + Реализации:
    - BasicCoefficientCalculators
    - DemographyCoefficientCalculators

UML диаграмма классов представлена на рисунке 3.1.

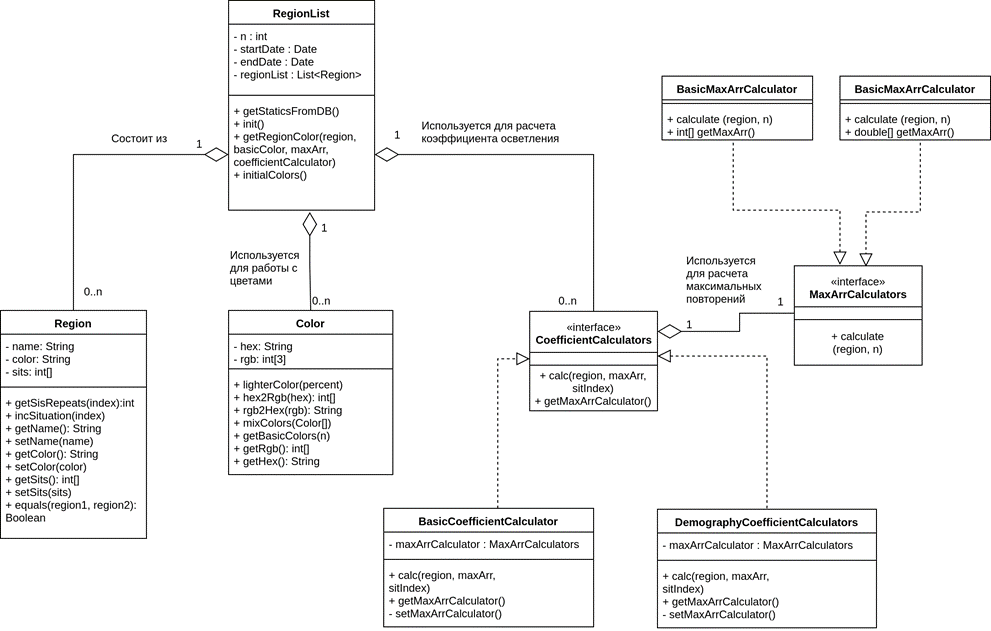


Рисунок 3.1. UML диаграмма основных классов модели

## Разработка взаимодействия с внешними API

## Для функционирования, программному средству необходима работа с внешними программными интерфейсами:

## API карт

## API для получения информации о регионе (координаты, популяция, если в локальной базе данных нет информации о данном регионе)

### Взаимодействие с API карт

Запрос на API карт отправляется тогда, когда необходимо создать карту либо нанести что-то на уже существующую карту.

Программное средство будет использовать следующие функции API карт:

* Создание карты (при загрузке страницы)

## Нанесение полигона с заданным цветом на карту

## Создание события при клике на заданный полигон

## Удаление всех полигоны из карты

## В качестве API для карт был использован Yandex Map API так как это довольно функциональный API с подробной документацией, причем в API присутствуют все необходимые функции [3].

### Взаимодействие с API Nominatim

## Для того, чтобы получать информацию о регионе в случае, если она отсутствует в базе данных, программное средство должно получать информацию из внешнего API и кешировать его в локальной базе данных.

## Программное средство будет использовать следующие функции данного API:

## Получение JSON с координатами заданного региона

## Получение JSON с информацией о регионе, в том числе информацией о популяции региона.

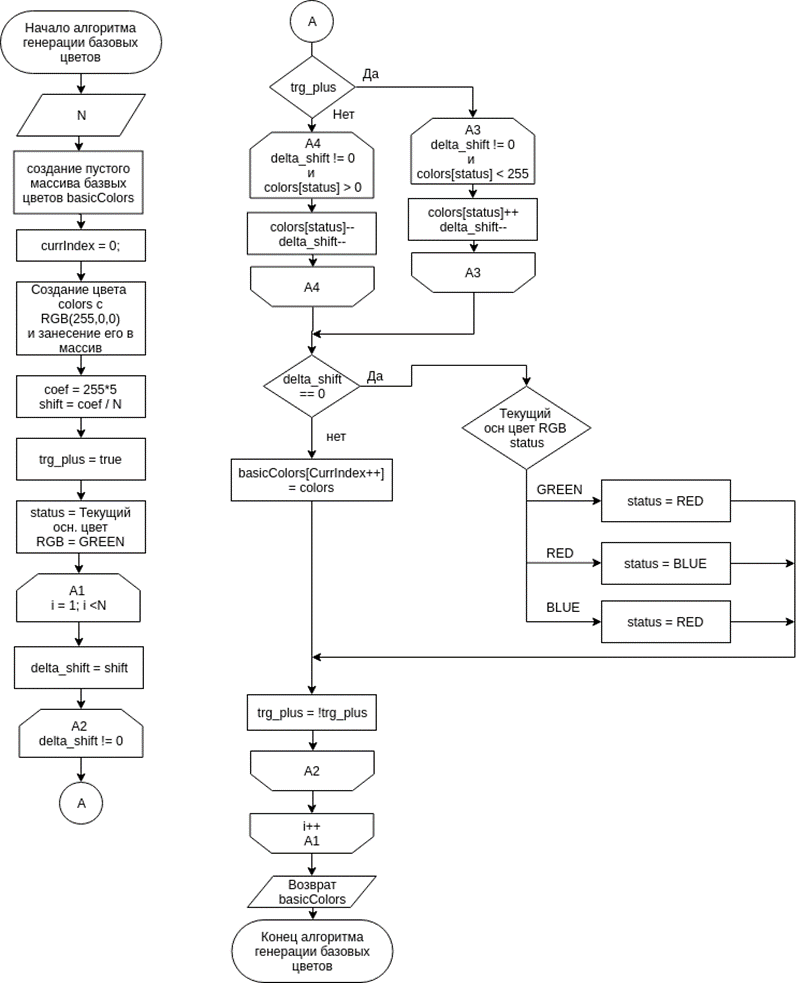
## В качестве такого API был выбран Nominatim API[4] так как это единственный свободный API с открытым исходным кодом (open source) API для таких целей и обладает всеми необходимыми возможностями

## Разработка алгоритма генерации базовых цветов

## Так как изначально у каждой ситуации региона должен быть свой цвет, необходимо разбить цветовой на N частей, где N – количество ситуаций. Таким образом у каждого региона получится свой цвет, причем цвета будут “пропорциональны”.

## Для этого цвета представляются в RGB и “отступ к следующей части цветового круга” получается путем манипуляций с r, g или b данного цвета.

## Схема алгоритма генерации базовых цветов представлена на рисунке 3.2



## Рисунок 3.2. Схема алгоритма генерации базовых цветов

## Разработка алгоритма осветления цветов

## Так как в зависимости интенсивности ситуации нужно будет осветлять базовые цвета, необходимо разработать алгоритм осветление цвета на определенное количество процентов.

## Приведенный на рисунке 3.3. алгоритм представляет собой осветление каждого из основных цветов (RGB) на необходимое количество процентов, что даст необходимый эффект для результирующего цвета.

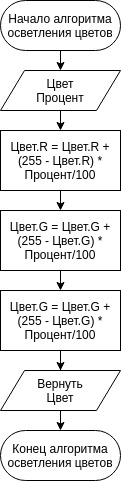


Рисунок 3.3. Схема алгоритма осветления цветов

## Разработка алгоритма смешения цветов

После получения осветленных цветов, необходимо произвести их смешение.

Приведенный на рисунке 3.4. алгоритм представляет собой получение результирующего цвета, основные цвета RGB которого равны среднему арифметическому основных цветов RGB смешиваемых цветов.

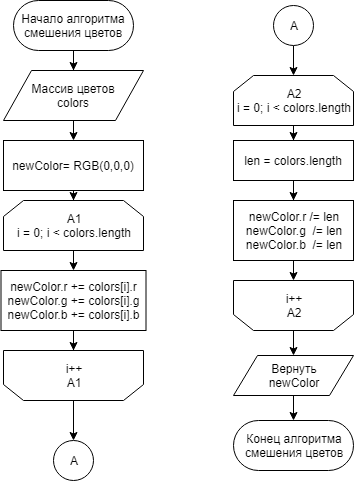


Рисунок 3.4. Схема алгоритма смешения цветов.

## Разработка алгоритма получения списка регионов

Для того, чтобы визуализировать информацию, надо ее предварительно обработать. Представленный на рисунке 3.5 алгоритм получает сырую статистику за заданный период, просчитывает количество повторений каждой ситуации в каждом регионе за этот период и исходя из этих данных осветляет базовый цвет каждой ситуации каждого региона в зависимости от режима и интенсивности повторения ситуации, после чего смешивает цвета ситуаций региона и получает результирующий цвет региона.

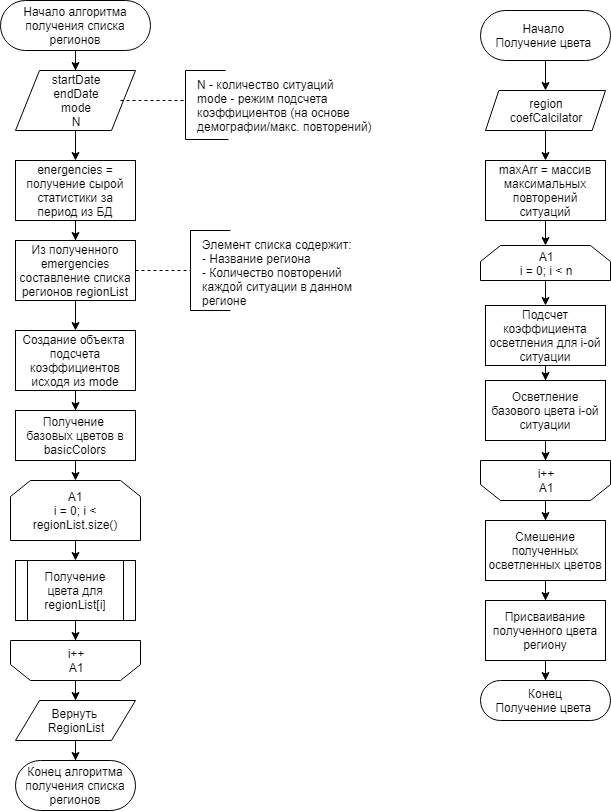


Рисунок 3.5. Схема алгоритма получения списка регионов.

## Разработка алгоритма подсчета коэффициента осветления цвета

Так как программное средство должно определять интенсивность повторения той или иной ситуации на основе общего числа повторений либо на основе количества ситуаций на душу населения по выбору пользователя, необходимо разработать два алгоритма:

* Алгоритм подсчета коэффициента осветления на основе общего числа повторений;
* Алгоритм подсчета коэффициента осветления на основе количества повторений ситуаций на душу населения.

### Разработка алгоритма подсчета коэффициента осветления на основе общего числа повторений

Алгоритм работает по следующему принципу: определяется максимальное количество повторений для каждой из ситуаций (Например, если есть два региона – А с повторениями (1, 2) и Б с повторениями (2, 1), то максимальное количество повторений – (2, 2)). После чего коэффициент для осветления цвета ситуации региона находится как процент, который составляет число повторений этой ситуации в этом регионе от максимального числа повторений для этой ситуации. Схема алгоритма приведена на рисунке 3.6.

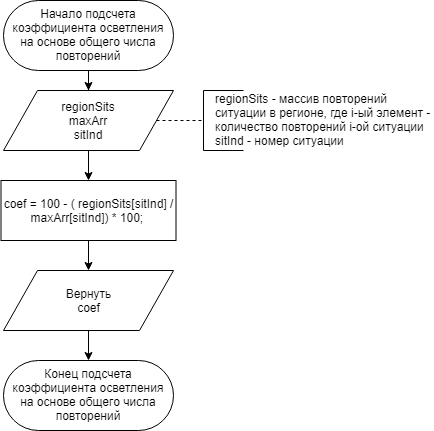


Рисунок 3.6. Схема алгоритма подсчета коэффициента осветления на основе общего числа повторений

### Разработка алгоритма подсчета коэффициента осветления на основе количества повторений ситуаций на душу населения.

Алгоритм работает по следующему принципу: определяется максимальное количество повторений на душу населения для каждой из ситуаций (Например, если есть два региона – А с числом жителей 10 и повторениями (1, 20) и Б с числом жителей 15 и повторениями (4, 30), то максимальное количество повторений – (0.26, 2)). После чего коэффициент для осветления цвета ситуации региона находится как процент, который составляет число повторений этой ситуации на единицу населения в этом регионе от максимального числа повторений на душу населения для этой ситуации. Схема алгоритма приведена на рисунке 3.7

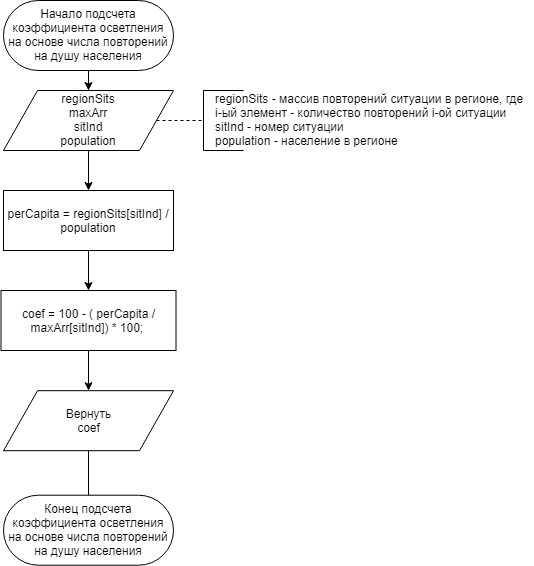
.

Рисунок 3.7. Схема алгоритма подсчета коэффициента осветления на основе числа повторений на душу населения

# КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Спецификация функциональных требований и спроектированная архитектура программного средства служат фундаментом, на котором основывается выбор наиболее подходящих технологий для разработки программного средства. Успешное и обоснованное завершение данных этапов позволит создать расширяемое, надежное и функциональное приложение, призванное решать поставленные задачи.

## Обоснование выбора средств разработки ПС

Для разработки программного средства были выбраны технология Java Spring и язык JavaScript, являющийся самым популярным для разработки клиентской части приложения.

**Технология REST**

REST (сокр. англ. Representational State Transfer, «передача состояния представления» или «передача репрезентативного состояния») – стиль построения архитектуры распределенного приложения. Был описан и популяризован в 2000 году Роем Филдингом, одним из создателей протокола HTTP. Самой известной системой, построенной в значительной степени по архитектуре REST, является современная Всемирная паутина[5].

Данные в REST должны передаваться в виде небольшого количества стандартных форматов (например, HTML, XML, JSON). Сетевой протокол (как и HTTP) должен поддерживать кэширование, не должен зависеть от сетевого слоя, не должен сохранять информацию о состоянии между парами «запрос-ответ».

Данная технология отвечает за различные реакции программного продукта на запросы к уровню промежуточного сервера.

**Язык программирования JavaScript**

JavaScript — это лёгкий, интерпретируемый, объектно-ориентированный язык с функциями первого класса, самый известный скриптовый язык для веб-страниц, но также используется во многих не браузерных окружениях. Прототипно-ориентированный, мультипарадигменный язык сценариев, который поддерживает динамический, объектно-ориентированный, императивный и функциональный стили программирования [6].

JavaScript запускается на стороне клиента Интернета, который может использоваться для создания/программирования того, как веб-страницы будут вести себя при наступлении каких-либо событий. JavaScript легко изучить, а также это мощный скриптовый язык, широко используемый для контролирования поведения веб-страниц.

JavaScript может функционировать и как процедурный, и как объектно-ориентированный язык. Объекты можно создавать программно во время выполнения, путем присоединения методов и свойств или пустых объектов во время выполнения, в отличие от синтаксических определений классов в компилируемых языках, таких как С++ или Java. После того, как объект был создан, он может быть использован в качестве плана (или прототипа) для создания похожих объектов.

**Технология AJAX**

Asynchronous Javascript and XML – подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее[7].

AJAX – не самостоятельная технология, а концепция использования нескольких смежных технологий.

Использование технологии динамического обращения к серверу «на лету», без перезагрузки всей страницы полностью.

**Язык программирования Java**

Java - сильно типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, с помощью виртуальной Java-машины.

Java - один из самых важных и широко применяемых языков программирования в мире на протяжении многих лет. В отличие от некоторых других языков программирования, влияние Java не только не уменьшилось со временем, а, наоборот, возросло. С момента первого выпуска он выдвинулся на передний край программирования приложений для Интернета. И каждая последующая версия лишь укрепляла эту позицию. Ныне Java по-прежнему остается первым и самым лучшим языком для разработки веб-ориентированных приложений. Проще говоря, большая часть современного кода написана на java. И это свидетельствует об особом значении языка jаvа для программирования [8].

Основная причина успеха java - его гибкость. Начиная с первой версии 1.0, этот язык непрерывно адаптируется к изменениям в среде программирования и подходам к написанию программ. А самое главное - он не просто следует тенденциям в программировании, а помогает их создать. Способность Java адаптироваться к быстрым изменениям в вычислительной технике служит основной причиной, по которой этот язык программирования продолжается оставаться столь успешным.

**База данных MySQL**

База данных MySQL — это самая популярная в мире база данных с открытым кодом. Благодаря своей проверенной производительности, надежности и простоте использования база данных MySQL наиболее часто используется для веб-приложений на таких ресурсах, как Facebook, Twitter, YouTube и все пять из пяти лучших веб-сайтов. Кроме того, ее особенно часто выбирают в качестве встроенной базы данных, распространяемой тысячами поставщиков программного обеспечения и производителей оборудования [9].

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

**Spring Framework**

Spring Framework обеспечивает комплексную модель разработки и конфигурации для современных бизнес-приложений на Java - на любых платформах. Ключевой элемент Spring - поддержка инфраструктуры на уровне приложения: основное внимание уделяется "водопроводу" бизнес-приложений, поэтому разработчики могут сосредоточиться на бизнес-логике без лишних настроек в зависимости от среды исполнения [10].

Возможности

* Внедрение зависимости
* Аспектно-ориентированное программирование, включая декларативное управление транзакциями
* Создание Spring MVC web-приложений и RESTful web-сервисов
* Начальная поддержка JDBC, JPA, JMS

**Инструмент сборки Maven**

Maven - это инструмент для сборки Java проекта: компиляции, создания jar, создания дистрибутива программы, генерации документации. Простые проекты можно собрать в командной строке. Если собирать большие проекты с командной строки, то команда для сборки будет очень длинной, поэтому её иногда записывают в bat/sh скрипт. Но такие скрипты зависят от платформы. Для того чтобы избавиться от этой зависимости и упростить написание скрипта используют инструменты для сборки проекта [11].

Основные преимущества Maven

* Независимость от OS. Сборка проекта происходит в любой операционной системе. Файл проекта один и тот же.
* Управление зависимостями. Редко какие проекты пишутся без использования сторонних библиотек(зависимостей). Эти сторонние библиотеки зачастую тоже в свою очередь используют библиотеки разных версий. Мавен позволяет управлять такими сложными зависимостями. Что позволяет разрешать конфликты версий и в случае необходимости легко переходить на новые версии библиотек.
* Возможна сборка из командной строки. Такое часто необходимо для автоматической сборки проекта на сервере (Continuous Integration).
* Хорошая интеграция со средами разработки. Основные среды разработки на java легко открывают проекты которые собираются c помощью maven. При этом зачастую проект настраивать не нужно - он сразу готов к дальнейшей разработке.
* Как следствие - если с проектом работают в разных средах разработки, то maven удобный способ хранения настроек. Настроечный файл среды разработки и для сборки один и тот же - меньше дублирования данных и соответственно ошибок.
* Декларативное описание проекта.

## Описание классов и методов

В программном средстве используется шаблон проектирования MVC, благодаря которому код, более понятный по структуре, логика и отображение разделены между собой. Дополнительно используются такие паттерны проектирования, как фабрика (Factory), репозиторий (Repository). Рассмотрим основные классы приложения.

### Описание класса Region

Класс Region представляет собой класс модели с информацией о конкретном регионе.

Класс содержит следующие свойства:

* name – название региона
* color – результирующий цвет региона
* sits – массив повторений ситуаций в регионе, где i-ый элемент массива это число повторений i-ой ситуации.

Таблица с основными методами данного класса приведена в таблице 4.1. (Геттеры/Сеттеры и конструкторы опущены)

Таблица 4.1. – основные методы класса Region

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Имя параметра** | **Назначение параметра** |
| incSituation | Увеличение на 1 количества повторений ситуации с нужным индексом | index | Индекс |
| equals | Сравнение двух регионов | region | Объект региона, с которым необходимо сравнить |

### Описание класса Color

Класс Color представляет собой класс модели с информацией о цвете.

Класс содержит следующие свойства:

* hex – hex представление цвета
* rgb – RGB представление цвета

Таблица с основными методами данного класса приведена в таблице 4.2. (Геттеры/Сеттеры и конструкторы опущены)

Таблица 4.2. – основные методы класса Color

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Имя параметра** | **Назначение параметра** |
| ligherColor | Осветляет цвет на необходимый процент и возвращает полученный объект (this) | percent | Процент |
| static hex2Rgb | Статический метод класса, переводящий цвет из hex в rgb | hex | Hex-представление цвета |
| static rgb2Hex | Статический метод класса, переводящий цвет из rgb в hex | rgb | RGB-представление цвета |
| static mixColors | Статический метод класса, смешивающий цвета и возвращающий результирующий цвет | colors | Массив цветов, которые необходимо смешать |
| static getBasicColors | Статический метод класса, возвращающий базовые цвета ситуаций | n | Количество ситуаций |

### Описание класса RegionsList

Класс RegionList представляет собой класс модели со сформированным списком регионов за заданный период, включая число повторений их ситуаций, их результирующий цвет.

Класс содержит следующие свойства:

* n – количество ситуаций
* startDate – дата, с которой начинается поиск записей в статистике
* endDate – дата, до которой начинается поиск записей в статистике
* mode – режим определения интенсивности (на основе демографии/общего числа повторений), используется в фабрике создания объекта подсчета коэффициента осветления
* coefficientsСalculators – Объект подсчета коэффициента осветления. Создается в фабрике на основе поля mode
* emergenciesRepository и populationsRepository – объекты репозиториев таблиц БД.

Таблица с основными методами данного класса приведена в таблице 4.3. (Геттеры/Сеттеры и конструкторы опущены)

Таблица 4.3. – основные методы класса RegionList

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Имя параметра** | **Назначение параметра** |
| getStaticsFromDB | Получает и возвращает статистику из базы данных | - | - |
| init | Инициализация списка ситуаций, который уже включает число повторений ситуаций | - | - |
| initialiseColors() | Создание объекта для расчета коэффициентов осветления через фабрику и инициализация цветов регионов | - | - |
| getRegionColor | Получает и возвращает цвет определенного региона (Цвет i-го региона получается путем осветление его базового цвета на процент,  на который он отличается от максимального показателя для этого региона) | region | Объект региона (класса Region) |
| basicColors | Массив базовых цветов |
| coefficientsСalculators | Объект для расчета коэффициента ответления цвета ситуации |

### Описание интерфейса CoefficientsСalculator

### Интерфейс содержит единственный метод – calc, который принимает регион и индекс ситуации.

### Присутствует две реализации интерфейса:

### BasicCoefficientsСalculator

### PopulationCoefficientsСalculator

### Каждая реализация содержит объект класса MaxArrCalculator для нахождения массива максимальный повторений.

Объекты этого интерфейса применяются для подсчета коэффициента осветления цвета ситуации.

### Описание интерфейса MaxArrCalculator

Интерфейс содержит единственный метод – calc, который принимает список регионов и количество ситуаций.

### Присутствует две реализации интерфейса:

### BasicMaxArrСalculator

### PopulationMaxArrСalculator

Каждая реализация содержит необходимые для себя репозитории, которые не передаются «вручную», а внедряются с помощью внедрения зависимостей (Dependency injection, DI).

### Описание класса HttpInteraction

Класс предназначен для отправки запросов на внешние url и имеет методы sendGet, sendPost, sendPut, sendDelete для отправки get, post, put и delete запросов соотвественно. Методы возвращают строку с возвращаемой сторонним сервером информацией. В качестве user-agent используется "Mozilla/5.0".

### Описание интерфейса APICommunication

Интерфейс описывает взаимодействие с внешними API для получения информации о регионе. Интерфейс содержит два метода getPopulation для получения информации о числе жителей региона по его названию и getCoords для получения координат региона по его названию.

Данный интерфейс имеет одну реализацию – NominatimAPICommunication с использованием Nominatim API, но имеется возможность в любой момент добавить реализацию для нового API.

Конфигурация для NominatimAPICommunication (url сервера API, ключ) хранятся в отельном файле в ресурсах. Взаимодействие с Nominatim API происходит строго по описанному в официальной документации[8] протоколу.

### Описание класса MainController

Данный класс является главным контроллером обращений к серверу. Он обрабатывает запросы на получение основных страниц: главной страницы по адресу “/” и страницей с документацией к API по адресу “/api”.

Основные методы данного класса описаны в таблице 4.4.

Таблица 4.4. – основные методы класса MainController

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Имя параметра** | **Назначение параметра** |
| index() | Возвращает главную страницу сервиса. | - | - |
| api() | Возвращает страницу с документацией по API с отредактированной моделью (в шаблон thymeleaf подставляет текущий хост) | request | Запрос пользователя |
| model | Модель, которая будет возвращена пользователю |

### Описание класса DBApiController

Данный класс является rest-контроллером для обработки rest-запросов клиента.

Класс содержит поля с объектами репозиториев таблиц базы данных и объекта логгера. Все эти объекты создаются и привязываются с помощью «внедрения зависимостей» (Dependency injection)

Основные методы данного класса описаны в таблице 4.5.

Таблица 4.5. – основные методы класса DBApiController

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Имя параметра** | **Назначение параметра** |
| getRegionsList() | Вовзращает JSON уже сформированного списка регионов с их результирующим цветом | startDate | Начальная дата диапазона в формате yyyy-MM-dd |
| endDate | Конечная дата диапазона в формате yyyy-MM-dd |
| mode | Режим отображения (базовый/на основе демографии. По умолчанию: basic) |

Таблица 4.5. – основные методы класса DBApiController

|  |  |  |  |
| --- | --- | --- | --- |
| getRegionsList | Вовзращает JSON уже сформированного списка регионов с их результирующим цветом | startDate | Начальная дата диапазона в формате yyyy-MM-dd |
| endDate | Конечная дата диапазона в формате yyyy-MM-dd |
| mode | Режим отображения (базовый/на основе демографии. По умолчанию: basic) |
| getRegionPopulation | Получение числа жителей в регионе | region | Название региона |
| getAllAccidents | Возвращает JSON со всеми записями статистики из базы данных по запросу | - | - |
| getByRegion | Возвращает JSON со всеми записями статистики из базы данных для нужного региона по | region | Название региона |
| getByDateRange | Возвращает JSON со всеми записями статистики из базы данных для нужного для нужного диапазона дат | startDate | Начальная дата диапазона в формате yyyy-MM-dd |
| endDate | Конечная дата диапазона в формате yyyy-MM-dd |
| getByDateRangeAndRegion | Возвращает JSON со всеми записями статистики из базы данных для нужного для нужного диапазона дат и нужного региона | startDate | Начальная дата диапазона в формате yyyy-MM-dd |
| endDate | Конечная дата диапазона в формате yyyy-MM-dd |
| region | Название региона |
| getSitsList | Возвращает JSON списка ситуаций (id, название) | - | - |
| getBasicColors | Возвращает JSON базовых цветов | - | - |
| getPolygon | Возвращает JSON полигона нужного региона | region | Название региона |
| getSitsCount | Возвращает количество ситуаций | - | - |

# ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

## Тестирование сервера с помощью модульного тестирования

Весь основной функционал сервера снабжен модульными тестами (unit-tests), что позволило автоматизировать данный процесс.

Юнит тесты покрывают следующий функционал:

* Перевод цвета из одного представления в другое
* Манипуляции с цветами
* Нахождение обработанного списка регионов
  + Нахождение цвета регионов
  + Подсчет количества регионов
  + Подсчет максимального количества повторений ситуаций

Юнит тесты проводятся как на корректных данные, так и «пустые» и некорректные данных.

## Тестирование целостной системы

Проведено тестирование целостной связки клиент-сервер. Целью данного испытания была проверка его работоспособности.

Установка и тестирование программного средства производилась на персональном компьютере с установленной операционной системой Ubuntu 18.04. ПС протестировано в последних версиях наиболее популярных браузеров: Google Chrome, Mozilla Firefox, Opera. Дополнительно работа сервера проверялась на персональном компьютере с установленной операционной системой Windows 10 и на выделенном сервере с установленной операционной системой Ubuntu 18.10 c 2 vCPU, 80 гб ПЗУ и 2 гб ОЗУ и выделенном сервере CentOS 7 с 1 vCPU, 20 гб ПЗУ и 1 гб ОЗУ. REST API сервиса был протестирован с помощью инструмента POSTMAN.

Набор тест-кейсов основного функционала главной страницы представлен в таблице 5.1.

Таблица 5.1 - тестирование функционала главной страницы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 0 | Отображение страницы | 1.Заход по адресу хоста / | Отрисовалась карта с данными диапазона по-умолчанию | Тест пройден! |
| 1 | Изменение диапазона дат | 1.Нажать на поле с датой  2. Изменение даты  3. Нажать «обновить» | Данные на карте обновятся | Тест пройден! |
| 2 | Изменение режима подсчета | 1.Нажать на combobox выбора режима  2.Выбрать режим «На основе демографии»  3. Нажать «обновить» | Данные на карте обновятся | Тест пройден! |
| 4 | Изменение диапазона дат на диапазон, которого нет в базе данных | 1. Нажать на поле с датой  2. Выбрать дату за 2000 год.  3. Нажать «обновить» | Будет отображена «пустая» карта | Тест Пройден! |
| 3 | Отображение информации о регионе | 1. Нажать на регион | Должна отобразиться информация о регионе, в том числе количество повторений каждой из ситуаций | Тест пройден! |

Набор тест-кейсов функционала API представлена в таблице 5.2.

Таблица 5.2 - тестирование функционала API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 0 | Отображение страницы с документацией к API | 1.Заход по адресу хоста /api | Отображение документации к API | Тест пройден! |

Таблица 5.2 - тестирование функционала API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | Проверка методов API с верными параметрами | 1.Зайти в POSTMAN  2. Ввести url метода API  3. Ввести корректные параметры  4. Сделать запрос | Отобразятся данные (их корректность проверена юнит-тестами на сервере) | Тест пройден! |
| 2 | Проверка методов API с неверными параметрами | 1.Зайти в POSTMAN  2. Ввести url метода API  3. Ввести некорректные параметры (например, некорректная строка)  4. Сделать запрос | Сообщение об ошибке | Тест пройден! |

Успешность прохождения тестов показывает корректность работы программы с реальными данными и соответствие функциональным требованиям.

# РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

## Серверная часть

Для корректной работы данного программного средства необходим сервер следующей минимальной конфигурации:

* OC: Linux / macOS 10.8.3 и выше / Windows 7 и выше
* Процессор: Pentium® III 800 МГц или AMD Athlon;
* RAM: 1 Гб;
* HDD: 1 Гб свободного места.

Дополнительно: наличие в операционной системе установленной СУБД MySQL, виртуальной машины JRE и инструмента для сборки maven.

До начала «развертывания» сервера, необходимо сконфигурировать базу данных. Для этого необходимо загрузить «дамп» пустой базы данных, находящийся в database/dump.sql. Сделать это можно следующей командой:

mysql -u имя\_пользователя -p -h имя\_сервера\_БД имя\_базы < database/dump.sql

После этого нужно заполнить главную таблицу со статистикой данными, после чего необходимо указать данные от пользователя БД в конфигурационном файле приложения src/main/resources/application.properties, настроив следующие свойства:

* spring.datasource.username
* spring.datasource.password

Опционально можно настроить порт, на котором будет работать сервер, добавив свойство server.port = ПОРТ.

После настроек конфигурации достаточно выполнить команду *mvn build* в каталоге с проектом и инструмент maven соберет jar со встроенным сервером tomcat в папку target. Последний шаг – запустить этой файл командой

*java –jar target/ИМЯ\_ФАЙЛА.jar*

Для того, чтобы сервер был доступен не только локально, потребуется дополнительная настройка и конфигурация apache/nginx, конфигурация которых не будет рассматриваться в рамках данной пояснительной записки.

## Клиентская часть

После захода на основной адрес клиентской части сервиса, появляется следующая страница, отображенная на рисунке 6.1.

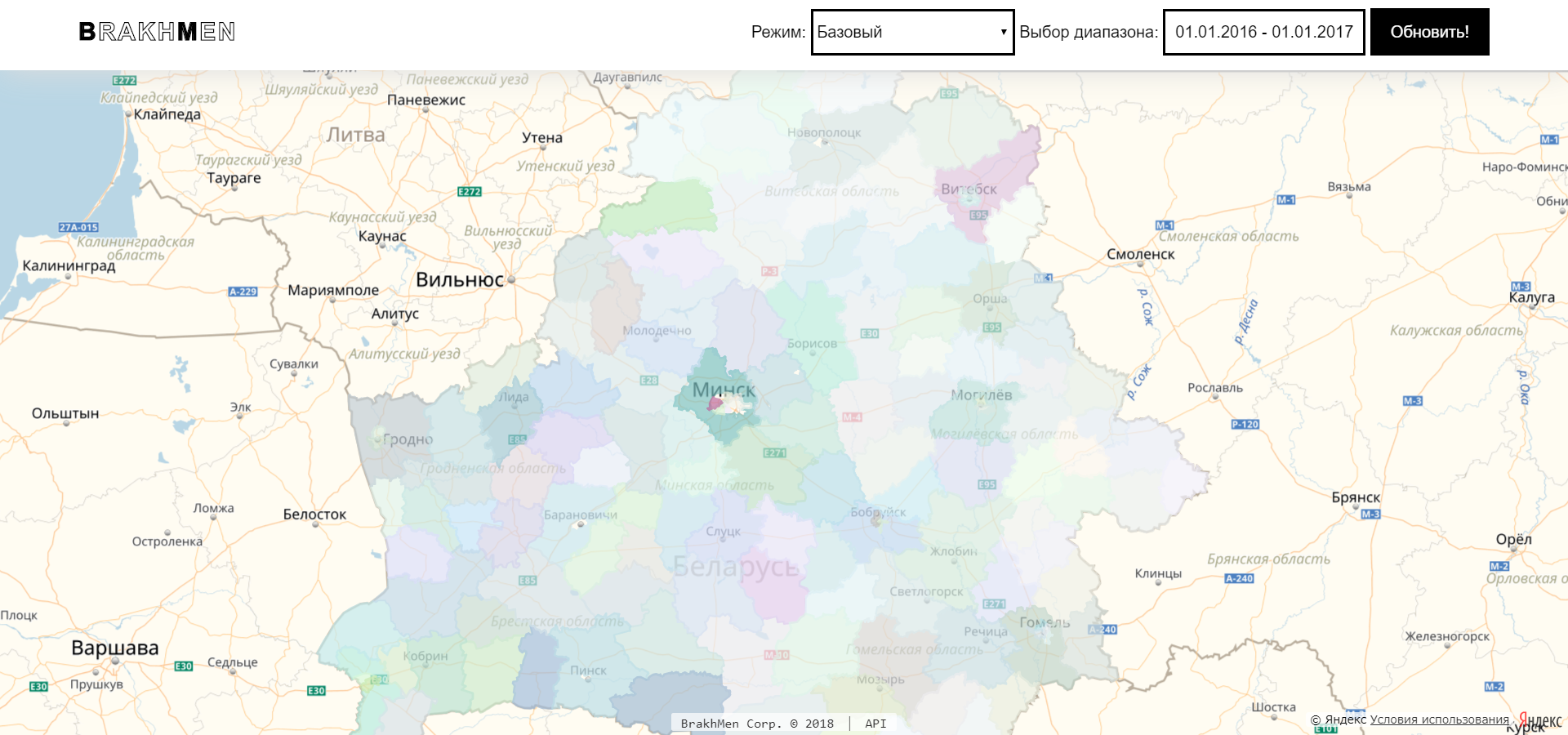


Рисунок 6.1. Скриншот основной страницы сервиса.

Чтобы изменить диапазон дат или режим подсчета коэффициентов, необходимо ввести нужные данные в полях, показанных на рисунке 6.2.

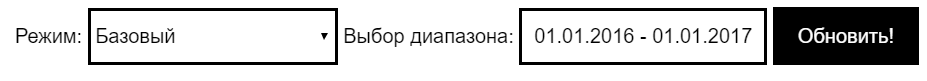


Рисунок 6.2. Скриншот полей для ввода дополнительных фильтров визуализации.

Если необходимо получить документацию по методам API, нужно нажать на ссылку «API» в футере сервиса, либо зайти по адресу /api на сервере. Скриншот страницы с документацией приведен на рисунке 6.3.

Чтобы посмотреть дополнительную информацию о конкретном регионе, необходимо произвести клик по этом региону, после чего появится окно с подробной информацией о количестве повторений ситуации, показанное на рисунке 6.4.



Рисунок 6.3. Скриншот с документацией к методам API сервиса.

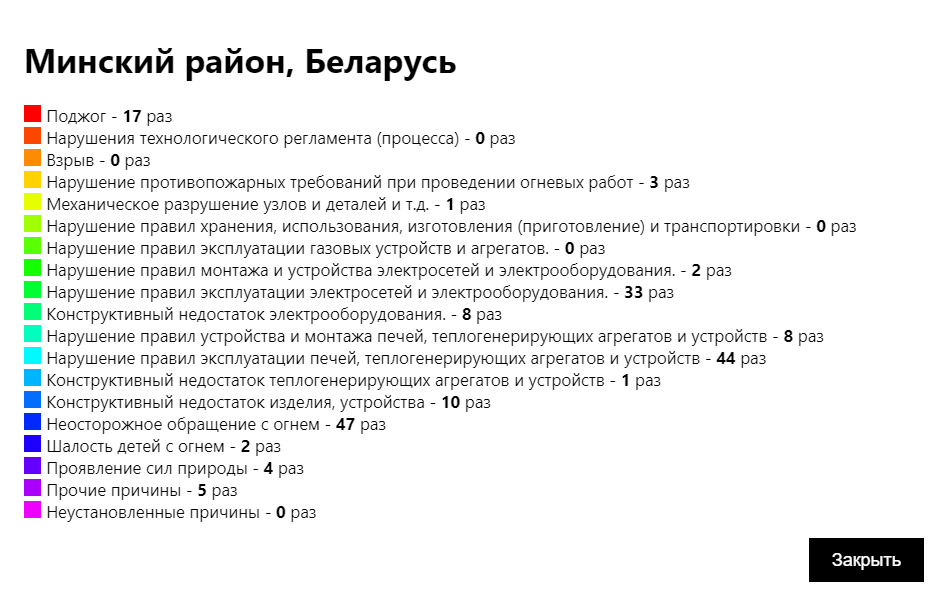


Рисунок 6.4. Скриншот окна с информацией о регионе.

Чтобы посмотреть описание конкретного метода API, необходимо кликнуть на этот метод, после чего раскроется информация о методе. Пример приведен на рисунке 6.5.

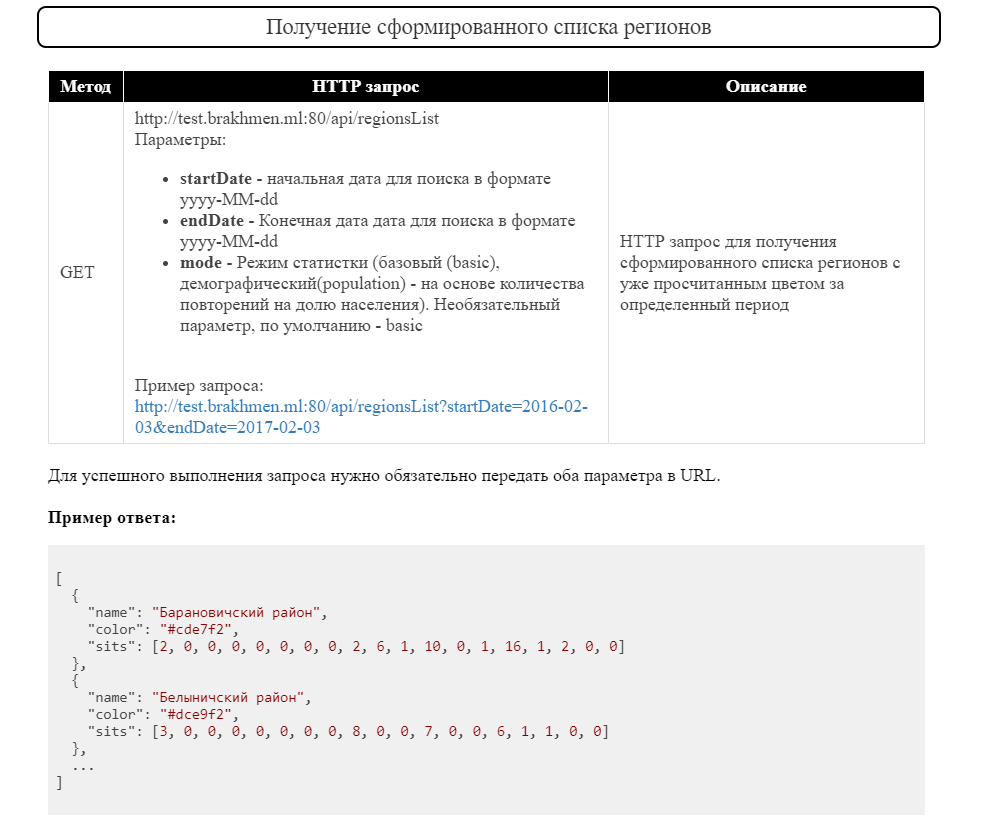


Рисунок 6.5. Пример описания метода API.

## API

Чтобы воспользоваться api сервиса, достаточно отправлять необходимые запросы на сервер, на котором размещено данное программное средство. Подробную документацию по методам API можно получить на соответственной странице сервиса.

Для примера, запрос на получение сформированного списка ситуаций за диапазон дат от 03.02.2016 по 03.02.2016 будет выглядеть следующим образом:

http://SERVER:PORT/api/regionsList?startDate=2016-02-03&endDate=2017-02-03

Пример ответа на подобный запрос будет выглядеть следующим образом:

[

{

"name": "Барановичский район",

"color": "#cde7f2",

"sits": [2, 0, 0, 0, 0, 0, 0, 0, 2, 6, 1, 10, 0, 1, 16, 1, 2, 0, 0]

},

{

"name": "Белыничский район",

"color": "#dce9f2",

"sits": [3, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 7, 0, 0, 6, 1, 1, 0, 0]

},

...

]

# ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта было разработано сервис, имеющий достаточный функционал для визуализации многомерных данных. Так же в сервисе присутствует REST API, благодаря которому любой разработчик может использовать сервис в своих целях.

При тестировании и отладке не было выявлено случаев некорректной работы программы, нестабильной работы, появления сторонних ошибок и т.д.

Простой и удобный пользовательский интерфейс с минимальным количеством кнопок позволяет пользователю легко привыкнуть программному средству и визуализировать данные с нужными фильтрами. Наличие документации по API в сервисе так же позволяет легко использовать программный интерфейс разработчикам.

Написанный код легко модифицируется для добавления новых фильтров, данных, для подключение другого постороннего API и т.д.

В процессе разработки, я изучил принципы разработки WEB сервисов с использованием языка Java и технологии Spring, принципы проектирование удобного REST API, научился работать с базой данных MySQL, изучил принципы построение клиентского приложения на языке JavaScript. Дополнительно я научился «поднимать» сервер на выделенной машине, используя nginx.

Разработанное программное средство можно применять для обработки статистики людьми, а также для презентаций какой-либо многомерной информации.

# СПИСОК ЛИТЕРАТУРЫ

[1] Noab Iliinsky. Designing Data Visualizations. - O’Reilly Media, 2011

[2] Панкратьев, А. С. Визуализация больших данных (BigData) / А. С. Панкратьев, Н. А. Пилинко, К. А. Голубев // Компьютерные системы и сети: материалы 54-й научной конференции аспирантов, магистрантов и студентов, Минск, 23 – 27 апреля 2018 г. / Белорусский государственный университет информатики и радиоэлектроники. – Минск, 2018. – С. 98 – 99.

[3] Документация - Yandex Maps API [Электронный ресурс] Режим доступа: https://tech.yandex.ru/maps/doc/jsapi/2.1/quick-start/index-docpage/. Дата доступа: 11.10.18

[4] Nominatim — OpenStreetMap Wiki [Электронный ресурс] Режим доступа: https://wiki.openstreetmap.org/wiki/Nominatim. Дата доступа: 15.10.18

[5] Архитектура REST / Хабр [Электронный ресурс] Режим доступа: https://habr.com/post/38730/. Дата доступа: 01.11.18

[6] О JavaScript - JavaScript | MDN [Электронный ресурс] Режим доступа: https://developer.mozilla.org/ru/docs/Web/JavaScript/О\_JavaScript. Дата доступа: 01.11.18

[7] AJAX - Национальная библиотека им. Н. Э. Баумана [Электронный ресурс] Режим доступа: https://ru.bmstu.wiki/AJAX#.D0.9F.D1.80.D0.B5.D0.B8.D0.BC.D1.83.D1.89.D0.B5.D1.81.D1.82.D0.B2.D0.B0. Дата доступа: 02.11.18

[8] Шилдт, Герберт. Java 8. Полное руководство; 9-е изд.: Пер. с англ. - М. : ООО "И.Д. Вильяме", 2015. - 1376 с. : ил. - Парал. тит. англ.

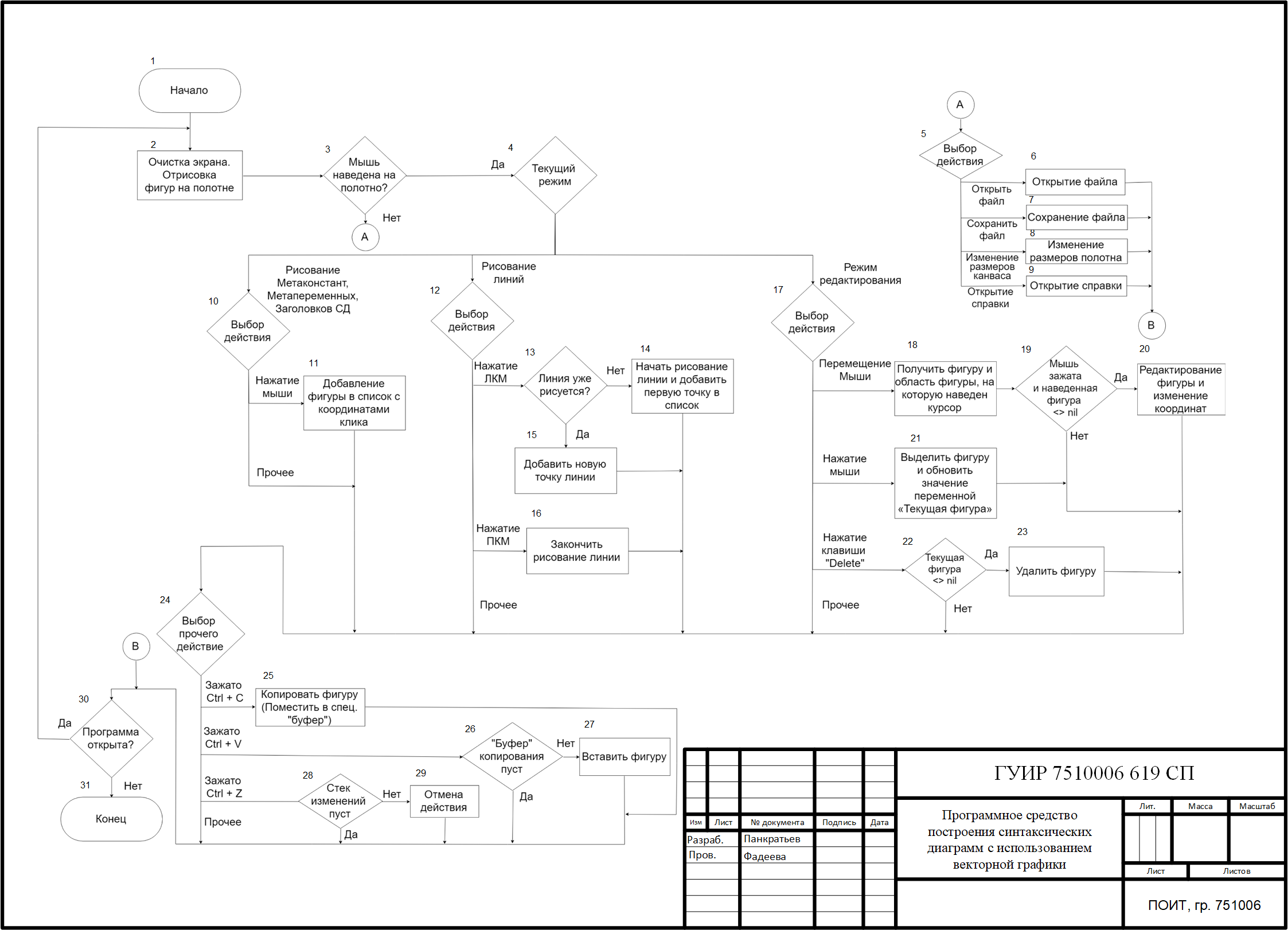
[9] MySQL | Самая популярная база данных с открытым исходным кодом | Oracle Россия и СНГ [Электронный ресурс]. Режим доступа: https://www.oracle.com/ru/mysql/. Дата доступа: 02.11.18

[10] Spring Framework | Spring по-русски! [Электронный ресурс]. Режим доступа: http://spring-projects.ru/projects/spring-framework/. Дата доступа: 02.11.18

[11] Руководство по Maven [Электронный ресурс]. Режим доступа: http://www.apache-maven.ru/. Дата доступа: 03.11.18

# ПРИЛОЖЕНИЕ 1

**Схема алгоритма работы программы**



# ПРИЛОЖЕНИЕ 2

## Текст программного модуля сервера

package men.brakh.emergencymap.controllers;

import men.brakh.emergencymap.BeansConfiguration;

import men.brakh.emergencymap.db.\*;

import men.brakh.emergencymap.dto.ColorDTO;

import men.brakh.emergencymap.http.APICommunication;

import men.brakh.emergencymap.models.Color;

import men.brakh.emergencymap.models.Population;

import men.brakh.emergencymap.models.Region;

import men.brakh.emergencymap.models.RegionsList;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.web.bind.annotation.\*;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping(path="/api")

public class DBAPIController {

@Autowired

private EmergenciesRepository emergenciesRepository;

@Autowired

private SituationsRepository situationsRepository;

@Autowired

private PolygonsRepository polygonsRepository;

@Autowired

private PopulationsRepository populationsRepository;

private static final Logger logger = LoggerFactory.getLogger(DBAPIController.class);

final SimpleDateFormat sdf1 = new SimpleDateFormat("yyyy-MM-dd");

/\*\*

\* Вовзращает JSON уже сформированного списка регионов с их результирующим цветом

\* @param startDate Начальная дата диапазона в формате yyyy-MM-dd

\* @param endDate Конечная дата диапазона в формате yyyy-MM-dd

\* @param mode Режим отображения (базовый/на основе демографии. По умолчанию: basic)

\* @return JSON уже сформированного списка регионов с их результирующим цветом

\*/

@GetMapping(value="/regionsList")

public List<Region> getRegionsList(@RequestParam String startDate,

@RequestParam String endDate,

@RequestParam(defaultValue = "basic") String mode) {

logger.info(String.format("Request to /regionList with params: %s, %s, %s", startDate, endDate, mode));

java.util.Date start;

java.util.Date end;

try {

start = sdf1.parse(startDate);

end = sdf1.parse(endDate);

} catch (ParseException e) {

logger.warn("Invalid date entered " + e.getMessage());

return new ArrayList<>();

}

java.sql.Date sqlStartDate = new java.sql.Date(start.getTime());

java.sql.Date sqlEndDate = new java.sql.Date(end.getTime());

RegionsList regionsList = new RegionsList(situationsRepository.getSitCount(), sqlStartDate, sqlEndDate, mode);

return regionsList.getList();

}

/\*\*

\* Получение числа жителей в регионе

\* @param region Регион

\* @return число жителей в регионе

\*/

@GetMapping(value="/population")

public long getRegionPopulation(@RequestParam String region) {

logger.info(String.format("Request to /population with params:%s", region));

return Population.get(region);

}

/\*\*

\* Возвращает JSON со всеми записями из базы данных по запросу

\* @return JSON со всеми записями из базы данных

\*/

@GetMapping(value="/accidents", params={})

public Iterable<Emergencies> getAllAccidents() {

logger.info(String.format("Request to /accidents with params: - "));

return emergenciesRepository.findAll();

}

/\*\*

\* Возвращает все записи из базы данных для нужного региона по запросу

\* @param region Название региона

\* @return Все записи из базы данных для нужного региона

\*/

@GetMapping(value="/accidents", params={"region"})

public Iterable<Emergencies> getByRegion(@RequestParam String region) {

logger.info(String.format("Request to /accidents with params: %s ", region));

Iterable<Emergencies> emergencies = emergenciesRepository.findByRegion(region);

for(Emergencies emergency : emergencies) {

System.out.println(emergency.getRegion());

}

return emergenciesRepository.findByRegion(region);

}

/\*\*

\* Возвращает все записи из базы данных за нужный диапазон дат по запросу

\* Отличается от getRegionsList тем, что getRegionsList возвращает уже подготовленный список регионов с их

\* цветом и тд, а getByDateRange - возвращает просто строки базы данных

\* @param startDate Начальная дата диапазона в формате yyyy-MM-dd

\* @param endDate Конечная дата диапазона в формате yyyy-MM-dd

\* @return Все записи из базы данных за нужный диапазон дат

\*/

@GetMapping(value="/accidents", params={"startDate", "endDate"})

public Iterable<Emergencies> getByDateRange(@RequestParam String startDate, @RequestParam String endDate) {

logger.info(String.format("Request to /accidents with params: %s, %s", startDate, endDate));

java.util.Date start = null;

java.util.Date end = null;

try {

start = sdf1.parse(startDate);

end = sdf1.parse(endDate);

} catch (ParseException e) {

logger.warn("Invalid date entered " + e.getMessage());

return null;

}

java.sql.Date sqlStartDate = new java.sql.Date(start.getTime());

java.sql.Date sqlEndDate = new java.sql.Date(end.getTime());

return emergenciesRepository.findByDateRange(sqlStartDate, sqlEndDate);

}

/\*\*

\* Возвращает все записи из базы данных за нужный диапазон дат для нужного региона

\* @param startDate Начальная дата диапазона в формате yyyy-MM-dd

\* @param endDate Конечная дата диапазона в формате yyyy-MM-dd

\* @param region Навание региона

\* @return Все записи из базы данных за нужный диапазон дат

\*/

@GetMapping(value="/accidents", params={"startDate", "endDate", "region"})

public Iterable<Emergencies> getByDateRangeAndRegion(@RequestParam String startDate, @RequestParam String endDate, String region) {

java.util.Date start = null;

java.util.Date end = null;

logger.info(String.format("Request to /accidents with params: %s, %s %s", startDate, endDate, region));

try {

start = sdf1.parse(startDate);

end = sdf1.parse(endDate);

} catch (ParseException e) {

logger.warn("Invalid date entered " + e.getMessage());

return null;

}

java.sql.Date sqlStartDate = new java.sql.Date(start.getTime());

java.sql.Date sqlEndDate = new java.sql.Date(end.getTime());

return emergenciesRepository.findByDateRangeAndRegion(sqlStartDate, sqlEndDate, region);

}

/\*\*

\* Возвращает список ситуаций (id, название)

\* @return JSON списска ситуаций [{"id": 1", "name": "..."}, ...]

\*/

@GetMapping("/sitsList")

public Iterable<Situations> getSitsList() {

logger.info(String.format("Request to /sitsList"));

return situationsRepository.findAll();

}

/\*\*

\* Возвращает базовые цвета

\* @return JSON с базовыми цветами

\*/

@GetMapping("/basicColors")

public ColorDTO[] getBasicColors() {

Color[] colors = Color.getBasicColors(situationsRepository.getSitCount());

ColorDTO[] colorWidthIds = new ColorDTO[colors.length];

for(int i = 0; i < colors.length; i++) {

colorWidthIds[i] = new ColorDTO(colors[i], i+1);

}

return colorWidthIds;

}

/\*\*

\* Получение полигона для региона

\* @param region Название региона

\* @return полигон региона

\*/

@GetMapping("/polygons/get")

public @ResponseBody String getPolygon(@RequestParam String region) {

region = region.replaceAll(" ", "%20"); // Заменяем пробелы на %20 ()

Polygons polygons = polygonsRepository.findFirstByRegion(region);

if(polygons != null) {

return polygons.getPolygon();

}

ApplicationContext context = new AnnotationConfigApplicationContext(BeansConfiguration.class);

APICommunication apiCommunication = (APICommunication) context.getBean("apiBean");

String coords = apiCommunication.getCoords(region);

Polygons n = new Polygons();

n.setRegion(region);

n.setPolygon(coords);

polygonsRepository.save(n);

logger.info(String.format("Added new row in table 'polygons'. Region: %s", region));

return coords;

}

/\*\*

\* Возвращаем количество уникальный ситуаций в БД

\* @return количество уникальных ситуаций в БД

\*/

@GetMapping("/sitsList/count")

public int getSitsCount(){

return situationsRepository.getSitCount();

}

}