МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)

Факультет информационных технологий
Кафедра «Инфокогнитивные технологии»
Образовательная программа (профиль):
«Корпоративные информационные системы»

Курсовая работа

по теме:

ВЕБ-ПРИЛОЖЕНИЕ ДЛЯ ВИЗУАЛИЗАЦИИ АНАЛИТИЧЕСКОЙ ИНФОРМАЦИИ ОБ ПОЖАРАХ В РЕГИОНАХ

Студент:
Группа № 231-362 *Р.Ф. Ишкильдин*

Предподаватель:
Старший преподаватель *Е.А. Харченко*

Предподаватель:
Старший преподаватель *М.В. Даньшина*

Москва 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Пожары, как природные, так и техногенные явления, представляют собой значительную угрозу для экосистем, человеческой жизни и экономического благополучия регионов. В условиях глобального изменения климата и увеличения антропогенной нагрузки на окружающую среду, частота и интенсивность пожаров возрастает. Эффективное управление рисками, связанными с пожарами, требует не только оперативного реагирования, но и глубокого анализа данных о произошедших инцидентах. Веб-приложения для визуализации аналитической информации об пожарах могут существенно повысить уровень осведомленности населения и органов власти о текущей ситуации, а также способствовать разработке стратегий по предотвращению и минимизации ущерба.

Несмотря на наличие различных систем мониторинга и управления пожарной безопасностью, многие из них страдают от недостаточной интеграции данных и отсутствия удобных инструментов для визуализации информации. Это приводит к затруднениям в интерпретации данных и принятию решений в условиях кризиса. Кроме того, существующие решения зачастую не учитывают специфические особенности различных регионов, что снижает их эффективность в контексте локальных условий.

На текущий момент существует несколько подходов к визуализации информации о пожарах, включая традиционные картографические системы и специализированные аналитические платформы. Однако такие системы часто требуют значительных ресурсов для разработки и поддержки, а также могут быть сложными в использовании для конечных пользователей. Другие решения включают использование бизнес-аналитики и специализированных программ для обработки данных, но они также имеют ограничения по доступности и удобству.

В данной курсовой работе будет представлено веб-приложение для визуализации аналитической информации о пожарах в регионах. Это позволит пользователям анализировать исторические данные для выявления закономерностей и тенденций. Ожидается, что разработанное решение повысит эффективность мониторинга и управления рисками, связанными с пожарами, а также улучшит информированность населения о потенциальных угрозах.

1 ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

1.1 Цель работы

Целью данной курсовой работы является разработка веб-приложения для визуализации аналитической информации о пожарах в различных регионах, что позволит пользователям анализировать и интерпретировать данные о пожарной активности.

1.2 Основные задачи работы

1. Анализ требований.
2. Сборка и подготовка данных.
3. Разработка серверной части.
4. Разработка клиентской части.
5. Документация и презентация результатов.

1.3 Исходные данные

Датасет представляет собой оперативные сведения МЧС России о географических точках, типах и датах природных пожаров, происходивших на территории России с 2012 по 2021 годы.
Источник: Инфраструктура научно-исследовательских данных. URL: https://data.rcsi.science/data-catalog/datasets/202/

1.4 Выбранные средства разработки

– Golang: язык программирования для разработки серверной части приложения
– JavaScript: язык программирования для создания клиентской части приложения
– Python: язык программирования для преобразования данных
– HTML/CSS: технологии для разметки и стилизации пользовательского интерфейса веб-приложения.

– MySQL: система управления базами данных для хранения информации о пожарах
– Nginx: используется как обратный прокси-сервер для перенаправления запросов от клиентской части к серверной

2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

2.1 Функциональные возможности

Веб-приложение для визуализации аналитической информации об пожарах в регионах представляет собой инструмент, предназначенный для сбора, обработки и представления данных о пожарах с целью повышения информированности населения и поддержки принятия решений в области пожарной безопасности. Данное приложение должно обеспечивать доступ к актуальной информации, а также предоставлять пользователям интуитивно понятный интерфейс для взаимодействия с данными.

2.1.1 Сбор данных и хранение данных

Важным элементом проектирования информационных систем является создание базы данных для хранения собранной информации. Эта база данных должна включать временные метки, географические координаты, типы пожаров и регион. Структурирование данных в такой форме обеспечивает их легкий доступ и анализ, что критически важно для оперативного реагирования на чрезвычайные ситуации.

Кроме того, сбор данных от пользователей через форму обратной связи является необходимым шагом для повышения эффективности системы. Внедрение функционала, позволяющего пользователям сообщать о пожарах или подозрительной активности в их регионе, должно быть реализовано через интуитивно понятный интерфейс на сайте. Пользователи смогут указывать местоположение, время происшествия и тип пожара, что дополнительно обогатит базу данных и повысит точность мониторинга.

2.1.2 Обработка и анализ данных

В рамках проектирования информационных систем необходимо интегрировать алгоритмы, способствующие глубокому анализу собранных данных. Это включает в себя применение статистических методов, направленных на выявление тенденций возникновения пожаров. В частности, исполь-

зование метода z-оценки позволяет стандартизировать данные и выявлять аномалии, что существенно повышает точность предсказаний и способствует более эффективному управлению рисками. Таким образом, комплексный подход к аналитическим инструментам обеспечивает не только обработку данных, но и их интерпретацию в контексте практических приложений для повышения безопасности.

2.1.3 Визуализация данных

Разработка визуализации, демонстрирующей статистические данные о пожарных инцидентах в различных регионах, включая количество зарегистрированных случаев, представляет собой важный аспект анализа и мониторинга пожарной безопасности.

2.1.4 Пользовательский интерфейс

В рамках проектирования информационных систем предусмотрена возможность создания учетных записей для пользователей. Данная функция обеспечивает индивидуальную идентификацию пользователей и их аутентификацию[1], что является необходимым условием для обеспечения безопасности и конфиденциальности данных.

Система также предоставляет пользователям возможность создания записей в базе данных. Эта функция позволяет пользователям вводить и сохранять информацию, что способствует эффективному управлению данными и их дальнейшему использованию в рамках системы.

2.1.5 Встраивание API приложения в другие системы

Разработка открытого API (интерфейса программирования приложений) представляет собой ключевой элемент для обеспечения интеграции с различными информационными системами[2]. Открытый API позволит осуществлять обмен данными между приложением и другими платформами, такими как системы управления чрезвычайными ситуациями, аналитические

[1] Соответствует пункту 5 на оценку хорошо

[2] Соответствует 3 пункту на оценку отлично

инструменты и мобильные приложения. Примером такой информационной системы является ”Система мониторинга и аварийного реагирования”. API будет поддерживать основные операции, включая получение информации о пожарах, добавление новых инцидентов и обновление существующих данных. Таким образом, создание данного API не только повысит функциональность приложения, но и обеспечит его совместимость с другими системами, что является важным аспектом в современном проектировании информационных систем.

2.2 Интерфейс

Веб-приложение, предназначенное для визуализации аналитической информации о пожарах в различных регионах, включает в себя многофункциональный интерфейс[3], который обеспечивает пользователям доступ к актуальным данным и инструментам для их анализа. Основная страница приложения служит центральной точкой взаимодействия, на которой расположена форма для генерации диаграмм, демонстрирующих статистику по пожарам.

2.2.1 Главная страница

Точкой начала взаимодействия с веб-приложением традиционно считается главная страница. Этот курсовой проект не исключение. Главная страница информационной системы представляет собой многофункциональный интерфейс, который включает в себя несколько ключевых компонентов. В первую очередь, она оснащена формой для формирования диаграмм, позволяющей пользователю выбирать тип диаграммы, а также указывать регион и дополнительные параметры, которые могут варьироваться в зависимости от выбранного типа диаграммы[4]. Эта функциональность обеспечивает гибкую настройку представления данных, что позволяет адаптировать его к конкретным аналитическим задачам. Кроме того, на странице предусмотрены элементы навигации, расположенные в верхней части интерфейса. Эти элементы обеспечивают доступ к функциям авторизации и регистрации пользователей, что значительно упрощает процесс входа в систему и способствует более

[3] Соответствие пункту 1 на оценку хорошо

[4] Соответствие пункту 3 на оценку хорошо

удобному взаимодействию с приложением. Таким образом, главная страница не только служит отправной точкой для анализа данных, но и обеспечивает интуитивно понятный доступ к основным функциям системы.

2.2.2 Страница регистрации

Данная страница веб-приложения представляет один из элементов пользовательского интерфейса, который включает в себя несколько важных функциональных компонентов. В первую очередь, на странице расположена форма регистрации, позволяющая пользователям создать новый аккаунт путем заполнения необходимых полей.

Кроме того, главная страница предоставляет навигационные элементы, которые обеспечивают доступ к Главной странице.

2.2.3 Страница входа

Форма для входа представляет собой механизм, который предоставляет зарегистрированным пользователям возможность доступа к Панели пользователя. Данная панель является центральным компонентом приложения, обеспечивающим пользователям возможность взаимодействия с базой данных. Также на странице присутствуют элементы навигации для перехода на Главную страницу.

2.2.4 Панель пользователя

Данная панель предоставляет возможность входящим пользователям добавлять новые записи в базу данных, что, в свою очередь, способствует актуализации информации и поддержанию её в соответствии с последними событиям. Как и на предыдущей странице здесь присутствует элементы навигации на главную страницу.

2.3 Структура проекта

В целях обеспечения гибкости и адаптивности разрабатываемой информационной системы было принято решение о разделении архитектуры

проекта на две независимые модуля: клиентскую часть (фронтэнд), предназначенную для взаимодействия с конечными пользователями, и серверную часть (бекенд), взаимодействие с базой данных[5]. Связь между этими модулями будет организована посредством прокси-сервера на базе Nginx. На диаграмме последовательности (см. Рисунок) можно увидеть порядок взаимодействия всеъ элементов структуры проекта.

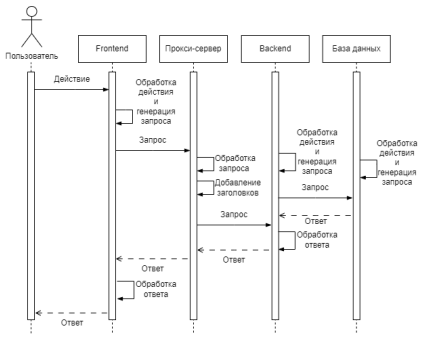

Рисунок 1 — Диаграмма последовательности

2.3.1 Бекэнд

В качестве языка программирования для реализации данного модуля был выбран язык Go. Это решение обусловлено несколькими факторами, включая простоту синтаксиса, высокую скорость выполнения и наличие обширного набора библиотек, предназначенных для серверной разработки. Модуль функционирует по следующему принципу: программа находится на по следующему адресу ”localhost:8080”. При обращении к необходимому адресу активируется соответствующая функция, которая, при необходимости, возвращает ответ пользователю.

2.3.2 Фронтэнд

Клиентская часть системы разработана с использованием языка программирования JavaScript, а также инструментов разметки и стилизации, таких как HTML и CSS. Выбор языка программирования обусловлен его обширными возможностями в области веб-программирования, что позволяет эффективно реализовывать интерактивные и динамичные веб-приложения. Данный модуль осуществляет обработку пользовательских действий и, при

[5]Соответствие пункту 2 на оценку отлично

необходимости, формирует и отправляет запросы к серверной части системы. Данный модуль расположен на домене ”localhost:5051”

2.3.3 Прокси-сервер

В связи с тем, что клиентские и серверные модули располагаются на различных доменах, возникают проблемы, связанные с механизмом CORS (Cross-Origin Resource Sharing). CORS представляет собой механизм, позволяющий веб-приложениям осуществлять запросы к ресурсам, находящимся на других доменах. Браузеры по умолчанию блокируют такие запросы из соображений безопасности, если сервер не предоставляет соответствующие заголовки, разрешающие кросс-доменные запросы. В качестве одного из наиболее эффективных решений данной проблемы рассматривается развертывание прокси-сервера. Для этой цели был выбран веб-сервер Nginx. Это решение обусловлено несколькими факторами, включая простоту настройки, надежность и высокую производительность.

Прокси-сервер обрабатывает входящий запрос от клиентского модуля и добавляет необходимые заловки, затем перенаправляет запрос на серверный модуль.

2.4 База данных

Проектирование и разработка базы данных началось с глубокого анализа предметной области, что является ключевым этапом в создании эффективной информационной системы.

В рамках данной предметной области выделяются следующие ключевые аспекты:

- Каждый пожар характеризуется множеством атрибутов, включая место (представленное географическими координатами и регионом), дату возникновения и описание типа пожара.
- Все записи о пожаре формируются пользователями системы.
- Каждый регион имеет атрибут — площадь.

– Пользовательская информация включает логин, пароль и дату регистрации.
– Пользователи системы разделяются на две основные роли: обычный пользователь и администратор.

2.4.1 Инфологическая модель

Знание предметной области позволило перейти к проектированию информационной модели базы данных (см. Рисунок 1).

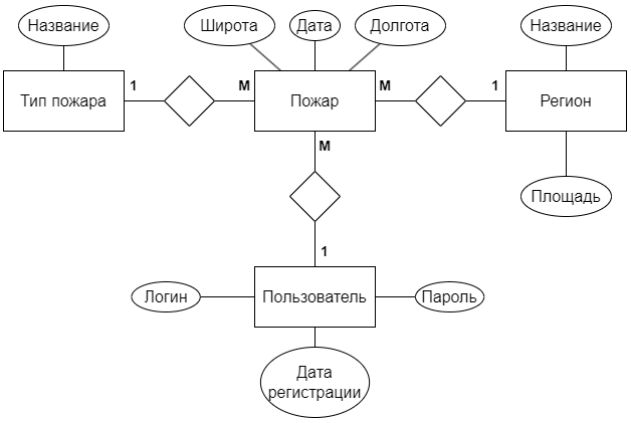

Рисунок 2 — Инфологическая модель

В процессе проектирования были выделены 4 сущности. А именно ”Тип пожара ”Регион ”Пользователь ”Пожар”.

”Тип пожара”связан исключительно с сущностью ”Пожар”. Отношение между этими сущностями представляет собой тип ”один ко многим так как каждый пожар может иметь только один тип. Сущность включает единственный атрибут — ”Наименование”.

Сущность ”Регион”также связана только с сущностью ”Пожар”. Здесь также наблюдается отношение ”один ко многим поскольку данные не учитывают межрегиональные пожары. Атрибуты включают ”Наименование”и ”Площадь”.

Сущность ”Пользователь”связана только с сущностью ”Пожар”. Тип связи — ”один ко многим так как у каждого пожара может быть только один пользователь, который его зарегистрировал. Атрибуты пользователя включают ”Логин ”Пароль”и ”Дату регистрации”.

2.4.2 Даталогическое проектирование

После завершения проектирования инфологической модели базы данных, был инициирован процесс создания даталогической модели. На первом этапе была разработана первичная реляционная модель, которая послужила основой для дальнейшего проектирования. (см. Рисунок 2).

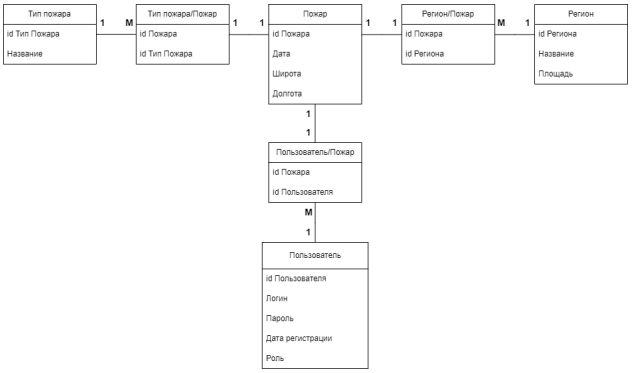

Рисунок 3 — Первичная реляционная модельь

В результате, была сформирована окончательная реляционная модель (см. Рисунок 3). в которой все отношения находятся в первой нормальной форме. Это обеспечивается тем, что значения всех атрибутов являются атомарными. Кроме того, данная модель соответствует нормальной форме Бойса-Кодда, поскольку все детерминанты функциональных зависимостей рассматриваемых отношений выступают в роли потенциальных ключей.

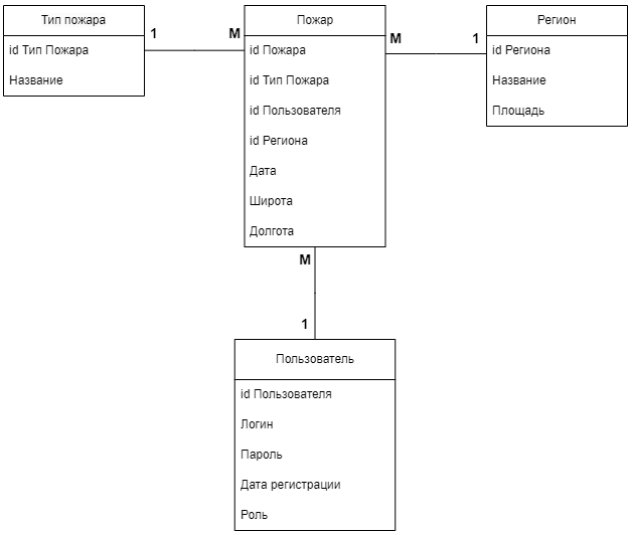

Рисунок 4 — Реляционная модельь

2.4.3 Физическая модель

Создание физической модели базы данных стало следующим этапом в процессе проектирования. В качестве системы управления базами данных (СУБД) была выбрана MySQL. Данный выбор обусловлен специфическими требованиями проекта: MySQL демонстрирует более высокую скорость обработки данных и производительность по сравнению с PostgreSQL, что является критически важным для достижения поставленных задач.

2.5 Подготовка данных

После подготовки физической модели базы данных, следует перейти к организации данных в структурированную форму. Анализ данных о пожарах представляет собой значимую задачу, требующую внимательной обработки и трансформации исходной информации для дальнейшей визуализации. В данном разделе рассматривается процесс преобразования данных о пожарах. Набор данных включает оперативные сведения МЧС России о географических координатах, типах и датах природных пожаров, зарегистрированных на территории России в период с 2012 по 2021 годы. Единицей наблюдения в данном датасете является место (географические координаты) пожара на конкретную дату. Для каждого наблюдения предоставляются следующие атрибуты: дата возгорания в формате ГГГГ-ММ-ДД, описание типа пожара, а также его долгота и широта. Процесс преобразования исходных данных можно разделить на несколько ключевых этапов. На данном этапе, как уже упоминалось ранее, был использован язык программирования Python, а также такие библиотеки, как Pandas, Numpy и ReverseGeocoder. Эти инструменты обеспечивают эффективную обработку и анализ данных, что является необходимым условием для создания качественной базы данных и последующей визуализации информации о природных пожарах.

2.5.1 Конвертация координат в регионы

В рамках первого подпункта были написаны несколько функций: get_coordinates() и to_regions(). Функция get_coordinates() загружает данные о координатах (широта и долгота) из CSV-файла:

1. Чтение данных с использованием read_csv для загрузки данных из указанного файла.
2. Создается пустой список arr, который будет использоваться для хранения координат.
3. С помощью функции iterrows() происходит итерация по каждой строке DataFrame. Для каждой строки значения широты и долготы преобразуются в тип float и добавляются в список arr в виде кортежа.
4. Возврат списка кортежей, содержащий все координаты.

Функция to_regions() выполняет основную работу по обратному геокодированию и сохранению результатов(см. таблицу 1):

1. Загрузка координат. Вызов функции get_coordinates().
2. Создается пустой словарь c, который будет содержать уникальные названия регионов. и их идентификаторов.
3. Обратное геокодирование.
    а) Для каждого элемента результата (el) проверяется код страны (el['cc']). Если он равен 'RU', это означает, что регион находится в России.
    б) Попытка добавить название региона (el['admin1']) в словарь c. Если название региона уже существует (вызывает исключение KeyError), то оно не добавляется повторно; если нет — присваивается новый идентификатор.
4. Создание DataFrame, который содержит уникальные названия регионов, на основе ключей словаря c.
5. Настройка индексов с помощью функции np.arange, чтобы они начинались с 1.
6. Сохранение результатов.

2.5.2 Сбор данных по площадям регионов

На основе открытых источников были собраны данные о площади регионов. Далее былы добавлена колонка, которая содержит соответствующие значения(см. Таблицу 2).

Таблица 1 — Таблица с регионами

region_id	region_name
1	Еврейская автономная область
2	Приморский край
3	Хабаровский край
4	Калининградская область
5	Амурская область
…	…

Таблица 2 — Таблица с регионами и с площадью

region_id	region_name	square
1	Еврейская автономная область	36271
2	Приморский край	165900
3	Хабаровский край	788600
4	Калининградская область	15000
5	Амурская область	361908
…	…	…

2.5.3 Получение типов пожаров

Третий этап включает создание файла() с типами пожаров и их идентификаторами. Это позволит упростить дальнейший анализ и визуализацию

1. Создание DataFrame, который содержит идентификаторы типов пожаров и их названия.
2. Извлечение уникальных пар значений с помощью функции drop_duplicates().
3. Сохранение результатов(см. Таблицу 3).

2.5.4 Создание итогового файла с данными

Этап создания итогового файла является завершающим в процессе обработки данных о пожарах. Он включает в себя объединение информации о термоточках (пожарах) с данными о регионах и их идентификаторами, что

Таблица 3 — Таблица с типами пожаров

type_id	type_name
1	Неконтролируемый пал
2	Торфяной пожар
3	Лесной пожар
4	Природный пожар
5	Контролируемый пал

позволяет получить структурированный набор данных для дальнейшего анализа и визуализации. Рассмотрим этот процесс более подробно.

1. Извлечение названий регионов из термоточек, которые были получены с использованием обратного геокодирования. Это было сделано в виду того, что при геокодировании получается регионы на английском.
2. Следующий шаг заключается в объединении полученных названий регионов с их идентификаторами. Для этого загружается файл regions.csv, который должен содержать соответствие между названиями регионов и их уникальными идентификаторами.
3. На этом этапе происходит загрузка изначального датасета из файла thermo.csv.
4. Объединение данных о изначального датасета и регионах.
5. Очистка итогового DataFrame. После объединения ненужные столбцы удаляются type_name и region, так как они могут не быть необходимыми для финального анализа или визуализации.
6. Обновление индексов, чтобы они начинались с 1.
7. Сохранение данных (см. Таблицу 4).

Таблица 4 — Таблица с типами пожаров

id	dt	type_id	lon	lat	region_id
1	2012-03-13	4	131.5866	47.8662	1
2	2012-03-13	4	131.5885	47.8809	1
3	2012-03-13	3	131.9871	48.4973	1
…	…	…	…	…	

3 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

3.1 Серверная часть

В целях повышения удобства разработки кодовая база данного модуля была структурирована на два файла. В файле main.go реализован функционал, отвечающий за обработку входящих запросов. Файл db.go содержит реализацию подключения к базе данных, а также функции, обеспечивающие взаимодействие с ней. Следует отметить, что из-за простоты архитектуры системы не возникла необходимость в использовании высоких уровней абстракции. Большая часть функционала системы была разработана с применением исключительно функционального подхода.

Для оптимизации процесса маршрутизации (см. Листинг 3.1) модуля был применён пакет gorilla-mux, который представляет собой маршрутизатор запросов и диспетчер для сопоставления входящих запросов с соответствующими обработчиками. Как было указано ранее, при поступлении запроса на домен, маршрутизатор, при наличии подходящего адреса, инициирует вызов заранее установленной функции. Аргументами таких функций выступают интерфейс, предназначенный для формирования HTTP-ответа, и указатель на структуру http.Request, используемую для представления HTTP-запроса. Также стоит заметить HTTP-сервер запускается на порту 8080 и использует созданный мультиплексор для обработки входящих запросов.

Листинг 3.1 — Маршрутизация запросов

```
1  func main() {
2      mux := http.NewServeMux()
3
4      mux.HandleFunc("/api/registration", createUser)
5      mux.HandleFunc("/api/verifyUser", loginValidation)
6      mux.HandleFunc("/api/getStandartScoreStat", getStandartStat)
7      mux.HandleFunc("/api/getRegions", getRegions)
8      mux.HandleFunc("/api/getFireTypes", getFireTypes)
9      mux.HandleFunc("/api/addData", addData)
10
11     http.ListenAndServe(":8080", mux)
12 }
```

Структура всех таких функций имеет общие черты. В качестве иллюстрации можно рассмотреть функцию getStandartStat() (см. Листинг 3.2), которая активируется при обращении к адресу ”localhost:8080/api/getStandartScoreStat”. Первым шагом в выполнении этой функции является вызов другой функции postParser(), которая принимает те же атрибуты, что и рассматриваемая функция. Как следует из названия, postParser() отвечает за парсинг POST-запросов и возвращает результаты в формате хэш-таблицы. Далее в процессе инициализируются два объекта: jsonData и err. Если количество пар ключ-значение превышает единицу, вызывается функция GetStandartStatWithTypes(formData), которая осуществляет запрос к базе данных и возвращает ответ в формате JSON вместе с возможной ошибкой. На следующем этапе происходит проверка на наличие ошибок; если переменная err не равна nil, выводится сообщение об ошибке, и выполнение функции завершается. В противном случае запрашиваемые данные выводятся в формате JSON.

Листинг 3.2 — Функция getStandartStat

```
1  func getStandartStat(w http.ResponseWriter, req *http.Request) {
2      formData := postParser(w, req)
3
4      var (
5          jsonData []byte
6          err      error
7      )
8
9      if len(formData) > 1 {
10         jsonData, err = GetStandartStatWithTypes(formData)
11     } else {
12         jsonData, err = GetStandartStat(formData)
13     }
14
15     if err != nil {
16         fmt.Fprintf(w, Возникла” ошибка: %s”, err)
17         return
18     }
19
20     w.Write(jsonData)
21 }
```

Далее стоит заострить внимание на работе с паролями, при создании и при валидации. Функция HashPassword() предназначена для хэширования паролей с использованием алгоритма bcrypt. Она принимает строку password в качестве входного параметра и возвращает хэшированный пароль в виде строки и возможную ошибку. Внутри функции происходит преобразование пароля в массив байтов, после чего вызывается метод GenerateFromPassword, который генерирует хэш с использованием стандартной стоимости вычислений (10). Если процесс хэширования завершается неудачно, функция возвращает пустую строку и ошибку, в противном случае она возвращает хэшированный пароль. Функция VerifyPassword принимает два аргумента: hashedPassword, который представляет собой хэшированный пароль, и password, который является паролем, который необходимо проверить. Она использует функцию bcrypt.CompareHashAndPassword для сравнения хэшированного пароля с введённым пользователем паролем. Если пароли совпадают, функция возвращает nil, что означает отсутствие ошибок, в противном случае она возвращает ошибку, указывающую на то, что пароли не совпадают.

Также необхоходимо осветить тему взаимодействия с базой данных. Для управления подключением к базе данных был релизован паттерн проектирования ”Singleton”. Это обеспечивает централизованное управление доступом к базе данных и способствует улучшению производительности модуля за счет оптимизации работы с соединениями. Была создана структура database, которая содержала одно поле connection, которое является указателем на объект типа *sql.DB. Этот объект предоставляет методы для работы с базами данных, включая выполнение запросов и управление транзакциями. Также в коде объявлены еще 2 переменные. Переменная instance используется для хранения единственного экземпляра структуры database, а переменная once из пакета sync обеспечивает потокобезопасное создание этого экземпляра. Функция getInstance отвечает за получение единственного экземпляра структуры database. Она использует метод Do из sync.Once, чтобы гарантировать, что код внутри анонимной функции выполнится только один раз. Внутри этой функции создается новый экземпляр структуры и устанавливается соединение с базой данных с помощью функции sql.Open. Если при открытии соединения возникает ошибка, экземпляр сбрасывается в nil, что позволяет избежать использования некорректного состояния.

Листинг 3.3 — Функция getStandartStat

```
type database struct {
    connection *sql.DB
}

var (
    instance *database
    once     sync.Once
)

func getInstance() (*database, error) {
    var err error
    once.Do(func() {
        instance = &database{}
        instance.connection, err = sql.Open("mysql",
            "root:@tcp(127.0.0.1:3306)/courseDB")
        if err != nil {
            instance = nil // Еслиошибка , сбрасываемэкземпляр
        }
    })
    return instance, err
}
```

Помимо функции, которая была разобрана в предыдущем абзаце. Большая часть функций в файле db.go имеет общую структуру. Как пример рассмотрим функцию GetStandartStat()[1]. Она предназначена для получения статистики по стандартным показателям из базы данных. Она принимает в качестве аргумента formData, который представляет собой хэш-таблицы, содержащую данные, необходимые для выполнения запроса. В начале функции происходит попытка получить экземпляр подключения к базе данных с помощью функции getInstance(). Если подключение не удалось установить (в случае ошибки), функция сразу возвращает nil и ошибку. Далее выполняется запрос с вызовом процедуры После успешного выполнения запроса создается срез[2] указателей на структуру StandartStatUnit, которая определяет поля для хранения данных о количестве пожаров, дате пожара и стандартном балле. В цикле for result.Next() происходит итерация по всем строкам результата запроса. Для каждой строки создается новый экземпляр структуры

[1]Соответсвует 4 и 6 пункту на отлично

[2]Срез - это расширенная реализация массива, которая поддерживает динамическое изменение размера.

StandartStatUnit, и данные считываются из результата запроса с помощью метода Scan(). Если при считывании возникает ошибка, функция возвращает nil и ошибку. Для более глубокого анализа следует рассмотреть SQL-процедуру, вызываемую в функции. На вход данной процедуры передается идентификатор региона. С помощью конструкции WITH формируется временная таблица, содержащая данные о дате, количестве пожаров, среднем значении и стандартном отклонении от среднего. Эта таблица группируется по дате. Затем из полученной таблицы осуществляется дополнительная выборка, в которой остаются первые два столбца, а также добавляется новый столбец, представляющий собой z-score, вычисляемый по следующей формуле(1),где

$$x$$

-количество пожаров в конкретную дату,

$$\mu$$

-стандартное отклонение,

$$\sigma$$

-среднее значение

$$z\_score = \frac{x - \mu}{\sigma}. \tag{1}$$

3.2 Клиентская часть

Описание этого модуля можно начать с верстки страниц. При написании css файлов, содержащих стилизацию страниц. Было отдельно прописан динамическое изменение элементов в зависимости от размеров экрана[3] Описание модуля продолжает с описания работы функций, которые обращаются к серверной части. Они преимущественно имеют общую структуру. В качестве примера можно рассмотреть функцию insertData()(см. Листинг 3.4). В начале функции происходит извлечение значений из HTML-элементов с помощью метода document.getElementById(). Создается объект XMLHttpRequest, который используется для выполнения асинхронного HTTP-запроса. Устанавливается метод и заголовок. После создается объект

[3]Соответствует 2 пункту на оценку хорошо

URL с необходимым адресом, далее передается. Параметры запроса формируются с помощью объекта URLSearchParams, куда добавляются все собранные данные, включая логин и пароль, которые извлекаются из локального хранилища браузера (localStorage). После отправляется запрос. Далее обрабатывается запрос и в зависмости от ответа выводит различные сообщения.

Рассмотрим механизм отрисовки диаграммы, который начинается с вызова функции getDrawChart(). В зависимости от типа диаграммы, осуществляется вызов соответствующей функции, ответственную за считывание необходимых данных и их передачу в функцию отрисовки, например, drawChart().

В рамках функции drawChart() происходит извлечение элемента canvas, а также установка различных параметров графика. Далее осуществляется отрисовка точек и линий, соединяющих эти точки. В случае, если z-оценка превышает 2, соответствующая точка выделяется красным цветом.

Данный метод использует функцию document.getElementById(), что позволяет получить доступ к HTML-элементу типа select. После этого инициируется запрос к серверной части системы, как было описано ранее. На этапе обработки ответа осуществляется итерация с использованием цикла foreach, который проходит по каждому элементу, полученному от сервера. В результате этого процесса создается элемент option, значение и текстовое содержимое которого формируются на основе полученных данных.

4 СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

4.1 Сценарий 1: Регистрация

1. Пользователь переходит на страницу регистрации.
2. Пользователь заполняет форму регистрации, включая поля: имя, пароль и подтверждение пароля.
3. Пользователь заполняет форму и нажимает кнопку ”Зарегистрироваться”.
4. Система проверяет корректность введенных данных.
5. При успешной регистрации пользователю отображается сообщение об успешной регистрации и предложение войти в систему.

4.2 Сценарий 2: Вход в систему

1. На главной странице пользователь нажимает на кнопку ”Вход”.
2. Система отображает форму для ввода логина и пароля.
3. Пользователь вводит адрес логин и пароль.
4. Пользователь нажимает кнопку ”Войти”.
5. Система проверяет введенные данные на корректность.
6. В случае успешной аутентификации пользователь перенаправляется на страницу страницу пользователя.

4.3 Сценарий 3: Генерация диаграммы

1. На главной странице пользователь выбирает тип диаграммы.
2. Система отображает форму дополнительные параметры в зависимости от типа диаграммы.
3. Пользователь нажимает кнопку ”Создать диаграмму”.
4. Система считывает данные.
5. Система делает запрос на серверную часть и получает данные. Отрисовывает диаграмму.

ЗАКЛЮЧЕНИЕ

В ходе разработки веб-приложения для визуализации аналитической информации о пожарах в регионах была достигнута основная цель — создание интуитивно понятного и функционального инструмента, способствующего повышению осведомленности и оперативности реагирования на чрезвычайные ситуации. Веб-приложение интегрирует современные технологии визуализации данных и предоставляет пользователям доступ к актуальной информации о состоянии пожароопасной ситуации в различных регионах. Анализ требований пользователей и исследование существующих решений позволили выявить ключевые функциональные возможности, графики динамики пожаров и возможность фильтрации данных по различным критериям. Эти функции обеспечивают не только удобство использования, но и высокую степень информативности представляемых данных. Кроме того, разработанное приложение учитывает аспекты безопасности и защиты данных, что является важным фактором в контексте работы с чувствительной информацией. Использование современных технологий веб-разработки обеспечило высокую производительность и масштабируемость системы. В заключение, разработанное веб-приложение является полезным инструментом для мониторинга и визуализации информации о пожарах. Оно может быть эффективно использовано как государственными учреждениями, так и частными организациями для повышения информированности и оперативности реагирования на пожароопасные ситуации. В дальнейшем планируется внести улучшения в функционал приложения на основе отзывов пользователей, что позволит сделать его более удобным и эффективным в использовании.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Network M.D.* HTML <canvas> element. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas. —.
2. *Rahul Y.* How to implement Singleton Design Pattern in Go. [Электронный ресурс] URL: https://dev.to/rahulyr/how-to-implement-singleton-design-pattern-in-go–5e1d. — 2020.
3. Метанит.ру. Язык программирования Go[Электронный ресурс]. - URL: https://metanit.com/go/. — Недра.