СОДЕРЖАНИЕ

[Введение 5](#_Toc198325173)

[1 Веб-приложение по усовершенствованию визуализации мониторинга качества воды 6](#_Toc198325174)

[1.1 Анализ предметной области 6](#_Toc198325175)

[1.2 Аналоги веб-приложения по усовершенствованию мониторинга качества воды в городе Минске 7](#_Toc198325176)

[1.3 Выводы и постановка задач на дипломное проектирование 11](#_Toc198325177)

[2 Разработка веб-приложения 14](#_Toc198325178)

[2.1 Программная структура веб-приложения 14](#_Toc198325179)

[2.2 Алгоритм работы программы 17](#_Toc198325180)

[2.3 Структура базы данных 27](#_Toc198325181)

[2.4 Выводы и оценка результатов разработки 30](#_Toc198325182)

[3 Анализ надежности веб-приложения 31](#_Toc198325183)

[3.1 Расчет надежности по модели сложности 31](#_Toc198325184)

[3.2 Расчет надежности по модели Джелинского – Моранды 37](#_Toc198325185)

[3.3 Расчет надежности по модели Муса 40](#_Toc198325186)

[4 Тестирование веб-приложения по усовершенствованию визуализации мониторинга качества воды посредством введения цветовых гексагонов на карте города Минска 43](#_Toc198325187)

[5 Технико-экономическое обоснование разработки и использования веб-приложения по усовершенствованию визуализации мониторинга качества воды посредством введения цветовых гексагонов на карте города   
Минска 55](#_Toc198325188)

[5.1 Характеристика разработанного веб-приложения по индивидуальному заказу 55](#_Toc198325189)

[5.2 Расчет основных затрат на разработку 56](#_Toc198325190)

[5.3 Расчет результата от разработки и реализации веб-приложения 60](#_Toc198325199)

[5.4 Расчет показателей экономической эффективности разработки и использования веб-приложения 63](#_Toc198325200)

[6 Охрана труда. Мероприятия по обеспечению высокой работоспособности и созданию комфортных условий труда при разработке и эксплуатации веб-приложения 67](#_Toc198325201)

[Заключение 72](#_Toc198325202)

[Список использованных источников 74](#_Toc198325203)

[Приложение А](#_Toc198325204) (обязательное) [Отчет о проверке на заимствование 77](#_Toc198325206)

[Приложение Б](#_Toc198325207) [(обязательное)](#_Toc198325208) [Листинг программы 78](#_Toc198325209)

[Ведомость дипломного проекта 103](#_Toc198325222)

# ВВЕДЕНИЕ

В современном мире качество питьевой воды остается одним из ключевых факторов, влияющих на здоровье населения. В Минске данные о качестве воды публикуются на сайте Минскводоканала в табличном формате, что затрудняет их оперативный анализ и визуальное восприятие [1].

Актуальность разработки обусловлена необходимостью улучшения способа визуализации информации о качестве воды для жителей Минска. Внедрение интерактивной карты с цветовой дифференциацией показателей позволит сделать данные более наглядными и доступными для широкой аудитории.

Анализ текущего решения, предоставляемого сайтом Минсководоканала, показал, что табличный формат требует знаний в соответствующей области, что затрудняет понимание о том, какое качество воды в доме. Это создает нишу для разработки специализированного приложения, которое сможет улучшить наглядность данных.

Цель дипломного проектирования — разработка веб-приложения для усовершенствованной визуализации данных о качестве воды в Минске на основе гексагональной тепловой карты. Приложение должно автоматически загружать данные с сайта Минскводоканала, анализировать их на соответствие санитарным нормам и отображать результаты в формате, который не требует знаний о предельно допустимых значениях.

Объектом исследования являются методы визуализации пространственных данных, таких как качество воды в определённом районе, с помощью гексагональных сеток и тепловых карт. Предмет исследования — разработка алгоритмов обработки и отображения данных о качестве воды с учетом их динамики и территориального распределения [2].

Задачи проекта включают проектирование серверной части приложения, отвечающую за автоматизацию сбора и обработки данных с сайта Минскводоканала, разработка алгоритма классификации качества воды по районам города оптимизацию запросов к *PostgreSQL*, реализацию интерактивной карты с гексагональной сеткой и тестирование на реальных данных. Каждый из этапов разработки направлен на создание системы, которая будет не только функциональной, но и удобной в использовании.[3]

Данный дипломный проект выполнен мной лично, проверен на заимствования, процент оригинальности составляет 75,37% (отчет о проверке на заимствования прилагается).

# 1 ВЕБ-ПРИЛОЖЕНИЕ ПО УСОВЕРШЕНСТВОВАНИЮ ВИЗУАЛИЗАЦИИ МОНИТОРИНГА КАЧЕСТВА ВОДЫ

## Анализ предметной области

В условиях роста урбанизации и увеличения нагрузки на городские инфраструктуры контроль качества питьевой воды становится одной из ключевых задач для обеспечения здоровья населения. Жители Минска, как и других крупных городов, нуждаются в достоверной и наглядной информации о состоянии водопроводной воды, чтобы своевременно реагировать на возможные отклонения от санитарных норм. В настоящее время данные о качестве воды публикуются на сайте Минскводоканала в табличном формате, что затрудняет их оперативный анализ и визуальное восприятие [4].

Разработка веб-приложения для усовершенствованной визуализации мониторинга качества воды направлена на решение этой проблемы. Приложение автоматически собирает данные с официальных источников, анализирует их на соответствие предельно допустимым концентрациям (ПДК) и отображает результаты в виде интерактивной тепловой карты с использованием гексагональной сетки. Такой подход позволяет пользователям быстро оценить ситуацию по всему городу, выявить районы с потенциальными рисками и отслеживать динамику изменений.

Для разработки такого веб-приложения можно выделить основные концептуальные требования:

1 Автоматизированный сбор данных: получение актуальных показателей качества воды с сайта Минскводоканала.

2 Анализ и классификация данных: сравнение значений с ПДК и присвоение цветовых меток в зависимости от уровня отклонения.

3 Визуализация данных на карте: отображение данных в виде гексагональной сетки, обеспечивающей равномерное покрытие территории и плавные цветовые переходы.

4 Исторический анализ: хранение и отображение данных за предыдущие периоды для выявления проблемных мест.

Целевая аудитория такого веб-приложения включает жителей Минска, которые смогут быстро оценить качество воды в своем районе и принять меры при необходимости, экологические организации и городские службы, которые смогут оперативно выявлять проблемные зоны.

Система состоит из трех основных структурных частей: веб-интерфейса, сервера для обработки собираемой информации и базы данных. Веб-интерфейс служит точкой входа в систему. Такое веб-приложение позволяет просматривать информацию о качестве воды не только в своей доме, но и в целом по городу Минску, позволяет сравнивать два временных промежутка между собой. Серверная часть системы отвечает за обработку данных, выполнение бизнес-логики и управление взаимодействием между веб-интерфейсом и базой данных. Она включает *API* для приема и отправки данных, алгоритмы расчета цвета для гексагона и сохранении информации в базу данных. *PostgreSQL* – выбранная база данных, которая обеспечивает хранение информации о гексагонах, времени расчета, параметрах воды на определенный момент времени.

Взаимодействие между модулями организовано следующим образом: Сервер по крону собирает данные о качестве воды в домах города Минска, после чего собирает преобразованную информацию в базу данных. Веб-интерфейс может запрашивать данные как по улице, так и по рассчитанному гексагону, что позволяет поддержать как нативный интерфейс (табличный формат), так и более продвинутое решение в виде гексагональной разметки [5].

Таким образом, система обеспечивает наглядное представление информации о качестве воды, упрощает ее анализ и способствует повышению осведомленности населения.

## 1.2 Аналоги веб-приложения по усовершенствованию мониторинга качества воды в городе Минске

Анализ существующих решений показал, что на рынке представлено несколько категорий систем, которые частично покрывают функционал. Однако ни одно из них не предлагает комплексного подхода, включающего как актуальные данные о качестве воды, так и отображении в виде тепловой карты. Рассмотрим наиболее значимые аналоги.

Сайт Минского водоканала — на сайте представлена информация о том, какие показания качества воды сейчас в конкретном доме. Расписаны технологии, по которым собираются данные, как часто эти данные обновляются, какие метрики используются для измерения параметров. Для получения качества воды в доме, необходимо вводить полный адрес в соответствующую поисковую строку, после чего на карте отобразиться отметка, где находится искомый адрес, а также под картой появятся показатели воды в табличном формате. Также на сайте представлена различная информация, связанная с источниками воды в городе Минске.

Основные достоинства включают простоту использования, поддержку не только веб-интерфейса для ПК, но и для мобильных устройств, доступность на белорусском рынке и слабовидящих людей. Интерфейс сайта интуитивно понятен, что позволяет пользователям быстро освоить его без дополнительного обучения. Однако, несмотря на свои преимущества, система не адаптирована для пользователей, которые хотят быстро увидеть ситуацию не только в своем доме, но и в соседних районах – необходимо вводить каждую улицу и дом по отдельности, что замедляет анализ текущей ситуации.

Интерфейс сайта Минского водоканала представлен на рисунке 1.1.

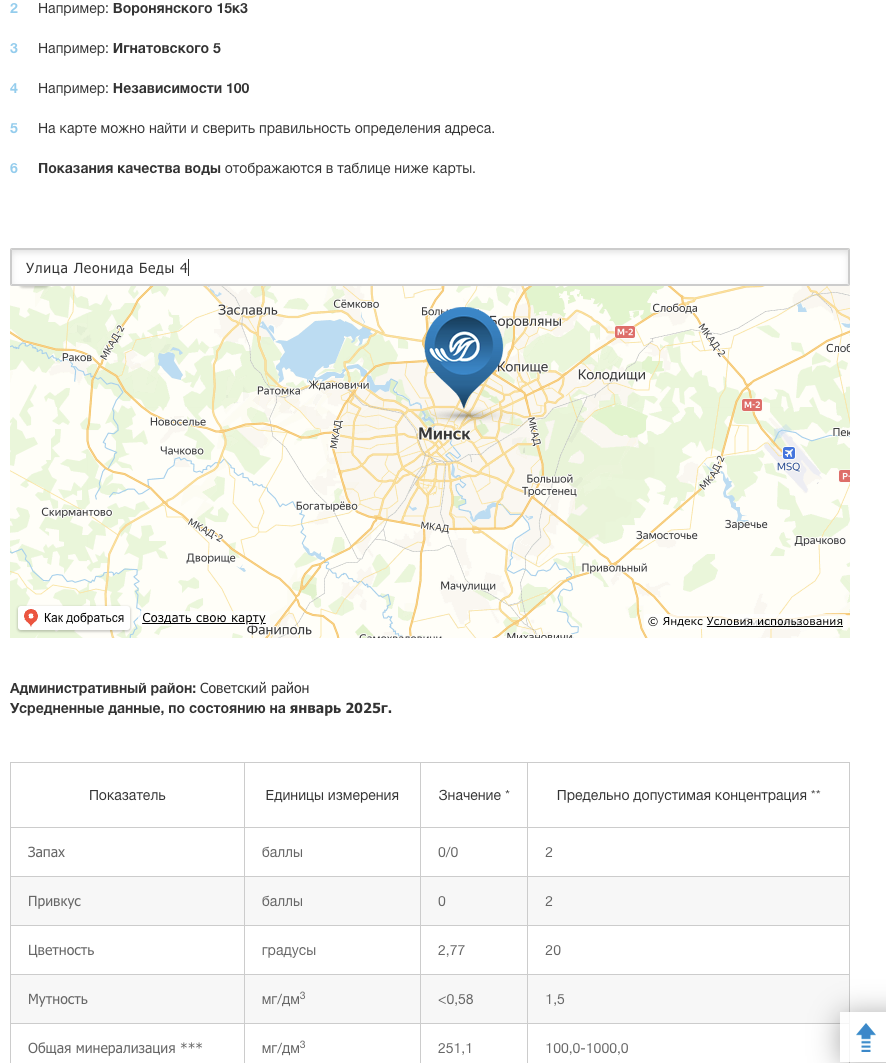


Рисунок 1.1 – Интерфейс сайта Минского водоканала

*Water* *Quality* *Map* — это сайт, который отображает информацию по различным показателям воды в реках, каналах, озерах, родниках, колодцах, скважин и водохранилищах в различных уголках России [6]. Преимущества такого сайта – это отметки на физической карте мира конкретных водохранилищ, что позволяет смотреть на ситуацию о качестве воды по более чем 15 параметрам в различных районах. Однако у сайта есть и недостатки, нет понимания того, что происходит в более мелких районах, а не только в водохранилищах, что может привести к пропуску какой-нибудь опасной зоны, в которой качество воды, например, могло ухудшится из-за техногенной катастрофы. Также минусом является неактуальность данных на этом сайте. По датам некоторых водохранилищ, информация не обновлялась уже более чем 3 года.

Для жителей Минска на таком сайте, к сожалению, нет полезной информации, так как ближайшее водохранилище находится довольно далеко от границ Беларуси.

Сайт также предоставляет возможность присылать данные из лабораторий или других аккредитованных организаций о качестве воды в определенном районе или водохранилище, что позволяет собирать данные не только от выбранного списка лабораторий, но еще и с пользователей, которые в этом заинтересованы. Так, например, если ранее эти карты не поддерживали белорусский регион, то теоретически такая поддержка возможна благодаря такому функционалу. Интерфейс сайта представлен на рисунке 1.2

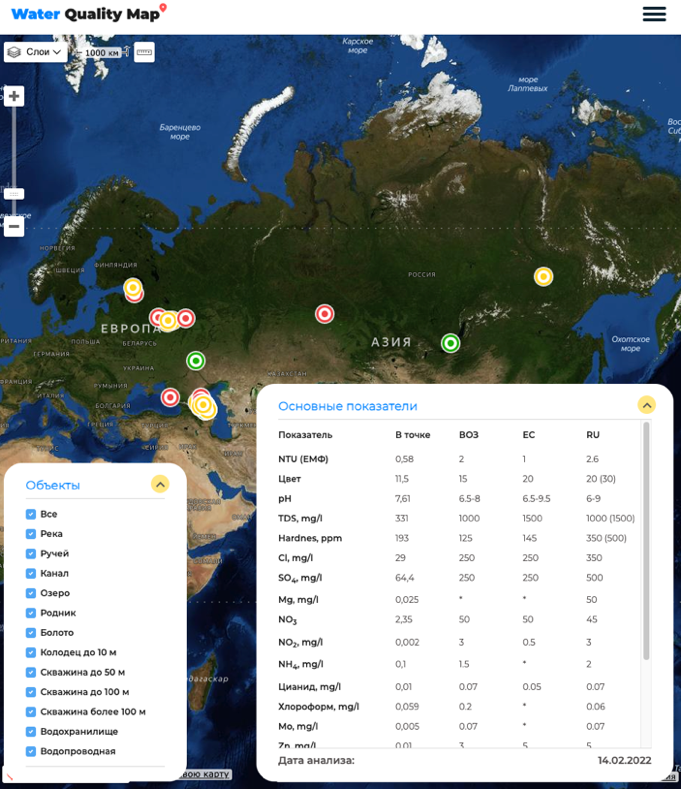


Рисунок 1.2 – Интерфейс сайта *WaterQualityMap*

*Wasserkiez* — это веб-приложение для визуализации водных контуров внутри стран европы. На карте отмечены водохранилища, пункты выдачи воды. Проект разработан поверх базы данных немецкой компании *TipTap*, которая занимается производством питьевой воды в Европе [7].

Этот сайт не подходит для анализа большого количества параметров воды, так как он предоставляет информацию только о местах, связанных с предоставлением питьевой воды. Интерфейс сайта приведен на рисунке 1.3.



Рисунок 1.3 – Интерфейс сайта *Wasserkiez*

*Uber* *H3* *Viewer* — это веб-приложение, которое позволяет по номеру гексагона и его резолюции, т.е. его размера, узнать его местоположение на карте. Имеет достаточно удобный и понятный интерфейс, но, к сожалению, не предоставляет публичное *API* для взаимодействия [8]. Интерфейс сайта с гексагональным покрытием карты представлен на рисунке 1.4.



Рисунок 1.4 – Интерфейса сайта *Uber* *H3* *Viwer*

Итог анализа аналогов показывает, что существующие системы частично покрывают необходимый функционал. Однако ни одно из решений не предлагает комплексного подхода, сочетающего и агрегацию данных о качестве воды, разметку на карте с помощью тепловой визуализации и интеграцию с картами. Все приведенные аналоги являются бесплатными решениями, а некоторые из них даже находятся в открытом доступе с исходным кодом под лицензией *MIT*. Разрабатываемая система будет выгодно отличаться доступностью для белорусского рынка (в частности, для жителей города Минска), низкой стоимостью внедрения и специализированным функционалом для сравнения параметров различных временных периодов.

## 1.3 Выводы и постановка задач на дипломное проектирование

Разработка веб-приложения для усовершенствованного мониторинга качества воды в г. Минске направлена на решение проблемы неудобного представления данных о состоянии водопроводной воды, которые в настоящее время публикуются в табличном формате на сайте Минскводоканала. Основная цель системы – обеспечить наглядную и интерактивную визуализацию данных о качестве воды с использованием гексагональной сетки на карте города, что позволит пользователям оперативно оценивать ситуацию, выявлять проблемные зоны и отслеживать динамику изменений.

Требования к разрабатываемой системе включают создание трех основных компонентов: веб-интерфейса приложения, серверной части для обработки данных и базы данных для хранения информации. Веб-приложение обеспечивает интеграцию с картами для визуализации гексагонов и качества воды, а также за возможность сравнения двух временных периодов. Серверная часть, реализованная на *Python*, отвечает за обработку данных, выполнение бизнес-логики и управление взаимодействием между веб-интерфейсом и базой данных. База данных на *PostgreSQL* обеспечивает надежное хранение информации о гексагонах и результатах аналитики.

Обмен данными между веб-интерфейсом и сервером осуществляется через *REST API*, что обеспечивает гибкость и масштабируемость системы.

Взаимодействие со смежными системами включает интеграцию с картографическими сервисами, такими как *Google Maps* или *Yandex Maps*. Также возможная интеграция с технологиями *Uber*, которые позволяют вычислять корректную разметку гексагонов на карте. В перспективе система может быть расширена за счет добавления оповещения пользователей о плохом качестве воды в их районе, для чего нужно будет поддержать систему пользователей, а также поддержать возможность отправлять уведомления на телефон и(или) почту.

Эргономика системы играет важную роль в ее успешном внедрении. Интерфейс приложения должен быть интуитивно понятным и удобным для использования гражданами Минска, а серверная часть — обеспечивать быстрый доступ к данным и формирование карты в удобном для пользователей формате.

Разрабатываемая система обладает рядом преимуществ. Во-первых, она предлагает специализированный функционал для анализа различных временных промежутков. Во-вторых, использование современных технологий, таких как *Python*, *JavaScript*, и *PostgreSQL*, позволяет минимизировать затраты на разработку и поддержку. В-третьих, модульная архитектура системы обеспечивает ее масштабируемость, что позволяет легко расширять функционал в будущем. Наконец, система разрабатывается с учетом потребностей белорусского рынка.

Выбор технологий для разработки системы обоснован их надежностью, производительностью и доступностью. *Python* обеспечивает высокую достаточную для целей веб-приложения производительность, быстроту разработки и безопасность серверной части, *JavaScript*, в связке с большим количеством фреймворков, позволяет создать удобный пользовательский интерфейс, а *PostgreSQL* — надежную и масштабируемую базу данных.

Основные задачи дипломного проектирования включают:

1. Изучение существующих способов сбора показателей воды. Это позволит понять какие показатели являются необходимыми для формирования цветовой разметки. Изучение предельно допустимых значение по каждому из показателей необходимо для того, чтобы правильно выбрать алгоритм формирования цвета у гексагона.
2. Разработка требований к функционалу приложения, включая веб-интерфейс, сервер для выполнения расчетов. Веб-интерфейс должен позволять пользователю получать информацию о качестве воды как при помощи гексагонов, нанесенных на полотно карты, но и при точечных запросах по конкретному адресу. Сервер веб-приложения должен уметь запускать периодические задачи для выполнения расчетов, уметь получать показатели воды для адресов и должен уметь агрегировать полученную информацию в гексагоны.
3. Проектирование архитектуры приложения с использованием *HTML, CSS, JavaScript* для реализации интерфейса веб-приложения, *Python и Flask* для реализации основной логики и *PostgreSQL* для хранения данных. Архитектура должна быть модульной, чтобы обеспечить гибкость и возможность расширения функционала в будущем.
4. Реализация основных модулей приложения, включая взаимодействие с картами, получение показателей воды по адресам и визуализацию тепловой карты с помощью гексагональной сетки. Особое внимание будет уделено визуализации качества воды, которая позволит пользователям легче оценивать ситуацию с качеством воды в городе Минске.
5. Организация хранения данных в базе данных *PostgreSQL* и обеспечение взаимодействия с ней. Это включает создание структуры базы данных для хранения информации об адресах и гексагонах, а также разработку механизмов для загрузки и обновления данных.
6. Тестирование приложения на реальных данных и подготовка документации. Тестирование позволит выявить и устранить ошибки, а документация – облегчит поддержку приложения в будущем.

Таким образом, дипломный проект направлен на создание специализированного решения, которое сочетает в себе простоту использования, аналитику текущей ситуации по городу, а также быстрый доступ к информации. Разрабатываемое приложение станет удобным инструментов для жителей Минска, помогая им быстро и удобно получать качество воды в своем районе или доме.

# 2 РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ

## 2.1 Программная структура веб-приложения

Разработка веб-приложения для визуализации мониторинга качества воды потребовала тщательного выбора современных технологий, обеспечивающих высокую производительность, удобство взаимодействия с пользователем и надежность хранения данных. Система построена по трехзвенной архитектуре, включающей фронтенд, бэкенд и базу данных, каждая часть которой реализована с использованием оптимальных технологических решений.

Для разработки пользовательского интерфейса были выбраны *HTML*, *CSS* и *JavaScript*. *JavaScript* – это богатая экосистема библиотек и инструментов, а также уже не первый год существование на рынке, что дает свои плюсы – большое количество материалов на просторах интернета.

Фронтенд часть системы также использует *JS API* Яндекс Карт. С помощью такой библиотеки можно получить интерактивное взаимодействие с картой (перемещение по карте, приближение или отдаление, выбор объектов на карте). Возможности у этого *API* довольно обширные, так как оно позволяет не только интерактивно взаимодействовать с картой, но и предоставляет возможность добавлять свои объекты на ее полотно. В нашем случае, такой функционал является ключевым, так как веб-приложение использует гексагоны для цветовой разметки. Для формирования гексагонов в интерфейсе используется информация из серверной части веб-приложения (список всех доступных на данный момент шестиугольников) и библиотека *H3*, которая позволяет находить координаты границ гексагона. С помощью полученной информации можно дополнительным слоем на карте разместить необходимые нам объекты с нужными *CSS* стилями [9, 10].

Серверная часть приложения разработана на *Python* с фреймворком *Flask* для взаимодействия по *REST* *API*. *Flask* был выбран вместо альтернатив (*Django, FastAPI*) из-за следующих преимуществ: *Flask* предоставляет только базовые компоненты для веб-сервера (маршрутизация, обработка запросов), что позволяет разрабатывать *API* именно так, как требуется для проекта, без избыточного функционала (*Django,* например, требует определенную структуру проекта; *FastApi –* проект не использует асинхронность, которая так важна для *realtime* задач – проект обновляет и пересобирает данные раз в сутки). Также среди используемых библиотек есть такие, как *Selenium*, для возможности обрабатывать данные с сайтов минского водоканала, а также *psycopg2* для взаимодействия с *PostgreSQL* [11, 12, 13].

Для хранения данных была выбрана реляционная база данных *PostgreSQL*. Этот выбор обусловлен ее надежностью, поддержкой сложных запросов и масштабируемостью. *PostgreSQL* позволяет эффективно хранить и обрабатывать большие объемы данных.

В процессе разработки использовались следующие инструменты: *Visual Studio Code*, в качестве основной среды разработки для написания кода на *HTML*, *CSS*, *JS*, *PyCharm*, в качестве среды разработки для *Python*, *WebStorm*, для взаимодействия с базой данных, *Git* для контроля версий и эффективного управления кодом.

Система состоит из нескольких модулей, которые взаимодействуют между собой через *REST API*. Это обеспечивает масштабируемость и модульность системы, позволяя легко добавлять новые функции или изменять существующие без необходимости переписывания всей системы. Основные модули включают:

1 Веб-интерфейс, предназначенный для пользователей, позволяющий взаимодействовать с картой и вводить адреса интересующих их мест. Пользователь может указать как улицу, так и номер интересующего гексагона. Также пользователь может сравнивать две даты по качеству воды.

2 Модуль обработки данных и формирования агрегируемых параметров по гексагону (серверная часть) отвечающий за обработку данных, полученных из минского водоканала. Сервер делает это в асинхронном режиме в качестве крон задачи, что позволяет реализовать более эффективный расчет данных на карте. Данный модуль реализован на *Python* с использованием фреймворка *Flask*, который обеспечивает гибкость и простоту разработки *API*. Сервер взаимодействует с базой данных *PostgreSQL*, где хранятся как текущие показатели качества воды, так и исторические данные. Также *PostgreSQL* используется сервером для сохранения рассчитанных средних показателей качества воды в гексагонах.

3 Модуль хранения данных (база данных) обеспечивает хранение всех данных, включая информацию о гексагонах, параметрах воды и результатах вычисления цвета гексагона. База данных построена на *PostgreSQL*, что обеспечивает надежность и масштабируемость.

Система веб-приложения для визуализации качества воды строится вокруг центрального модуля обработки данных. Серверная часть отвечает за подсчет средних значений по гексагону. Эта логика специально отселена в отдельную компоненту – крон таску, которая рассчитывает значения для всех гексагонов и которая реализована с помощью библиотеки *Schedule*. *Schedule* – это технология, которая позволяет периодически запускать действие, указанное в специальных функциях. Так, например, в нашем случае мы запускаем периодически тяжелые расчеты (сбор информации о всех улицах города Минска, параметрах воды в домах и агрегирование собранной информации в гексагонах), после чего сохраняем всю полезную информацию внутрь базы данных. Период выполнения задач по подсчету показателей воды составляет сутки, а для сбора информации по адресам города Минска – неделя. Тут стоит отметить, что из-за асинхронной схемы взаимодействия между частью с расчетами и потенциальными клиентами сервиса могут возникать повторные расчеты – если выполнение задачи по крону еще не завершено, а пользователь уже хочет получить данные [14].

Взаимодействие между модулями организовано следующим образом: сервер в асинхронном режиме рассчитывает полезные для всей системы данные – это параметры воды по гексагонам. Для выполнения этой цели серверу необходимо получить список всех улиц и всех домов города Минска, после чего рассчитать средние показатели качества воды в конкретном гексагоне. Веб-интерфейс отправляет данные в сервер для извлечения информации из *PostgreSQL*, которая предварительно была посчитана с помощью асинхронной задачи. При отсутствии полезной информации в *PostgreSQL* (например, если задача была временно отключена или выполняется слишком долго) происходит синхронной расчет информации, запрошенной пользователем.

Пользователи взаимодействуют с системой через интерактивную карту, где могут кликать на цветные гексагоны для просмотра детальных показателей качества воды во всплывающих окнах. В интерфейсе есть настройки отображения данных по временным периодам. Пользователи могут масштабировать карту для детального изучения конкретных районов или просматривать общую картину по всему городу. Дополнительная панель инструментов позволяет переключаться между различными режимами визуализации, включая тепловую карту и сравнительный анализ по месяцам. Все данные обновляются автоматически при изменении информации в источнике, обеспечивая актуальность представленной информации.

Таким образом, система обеспечивает комплексный подход визуализации качества воды. Взаимодействие между модулями через *REST API* обеспечивает гибкость и масштабируемость системы, что позволяет легко адаптировать ее под нужды конкретных пользователей или организаций.

## 2.2 Алгоритм работы программы

Процесс работы веб-приложения можно разделить на две основные части: это алгоритм работы интерфейса веб-приложения и его серверной части.

Процесс работы серверной части начинается с запуска исполняемого файла, при котором происходит инициализация всех необходимых компонентов системы. На этом этапе инициализируются настройки для внешних хранилищ, прописываются *API* ключи для внешних источник, создаются *web*-драйвер для работы с библиотекой *selenium*. На этом этапе проверяется доступность базы данных. При запуске приложения также инициализируются компоненты, которые отвечают за функционирование *REST API,* и компоненты, которые отвечают за запуск периодических задач.

Алгоритм запуска серверной части веб-приложения включает следующие основные шаги:

1 Инициализация компоненты *Flask.* При старте приложения регистрируются обработчики событий, которые поступают на сервер. Таких обработчиков запросов с фронтенда у сервера шесть: получение информации об адресе, получение информации о гексагоне, получение информации о всех адресах, получение информации о всех гексагонах с необходимой резолюцией, получение цветов гексагонов. Также по окончанию инициализации для всех обработчиков в ответе сервиса добавляются хедера, которые содержат информацию для *Cross-Origin Resource Sharing* [15].

2 Регистрация обработчиков регулярных событий. Выставление периодов регулярных событий. Регистрация обработчиков выполняется при помощи библиотеки *Schedule.* Периоды обновления равны: сутки, для обновления информации о показателях воды в домах города Минска и для обновления информации о средних показателях воды для гексагонов, и неделя для обновления информации о списке всех домов города Минска [16].

3 Установление соединения с базой данных. Такое соединение полезно устанавливать в начале работы веб-приложения, чтобы на этапе запуска провалидировать все необходимые настройки доступов для получения данных из *PostgreSQL.*

4 Инициализация конфигов для работы серверной части. В этот пункт можно отнести конфиги для работы с геокодированием, лимитами на запросы во внешние источники, конфиги по настройки резолюций, отображаемых на карте города Минска.

5 Запуск регулярных расчетов. После этого этапа сервер начинает обрабатывать информацию об адресах города Минска, собирать средние показатели воды по гексагонам и обновлять информацию об актуальном списке адресов.

1. По завершению запущенного регулярного расчета, cсервер переходит в режим ожидания. В таком режиме сервер может обрабатывать запросы либо выполнять регулярные задачи, если со времени запуска прошлой задачи прошло необходимое количество времени.

Таким образом, после запуска сервера, клиенты веб-приложения могут получить информацию из сервера. На рисунке 2.1 приведен пример логов сервера веб-приложения, из которых можно понять, что первая задача была успешно завершена и сервер работает в режиме ожидания.

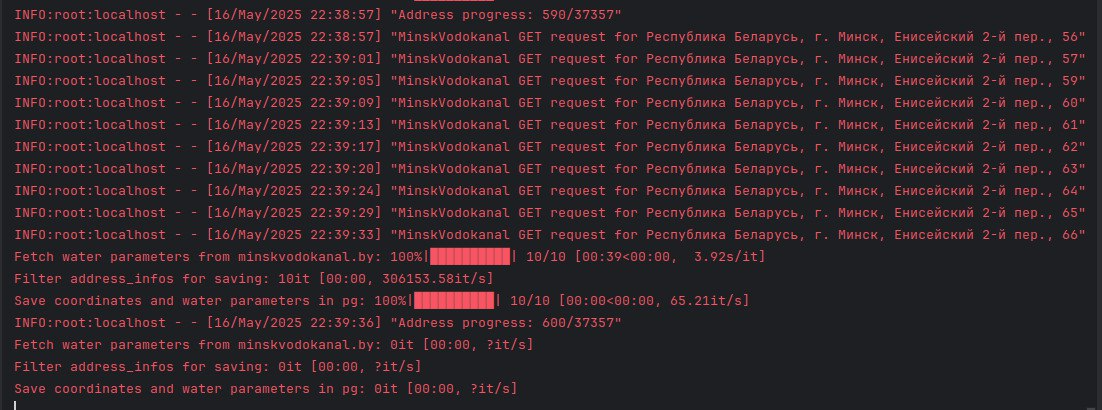


Рисунок 2.1 – Инициализация работы сервера веб-приложения

После запуска сервера веб-приложения и загрузки всех необходимых для этого компонент сервер, как и было сказано выше, переходит в режим ожидания для приема входящих запросов и расчета регулярных задач. Ядром приложения выступает схема асинхронного расчета и агрегации параметров воды для всех гексагонов города Минска. Во время выполнения задачи, сервер запрашивает данные из внешних источников для получения всех домов города Минска, а также для получения параметров воды по адресам.

Алгоритм работы сервера для получения информации о жилых домах города Минска:

1 Сервер по истечению *ttl,* равному 1 недели с прошлого запуска, начинает извлечение данных об жилых домах города Минска из внешних источников. Внешний источник для этого случая, в стандартном поведении программы, является сайт *ato.by*,на котором собрана информация не только о полном адресе жилого дома, но и приведено его местоположение на карте.

2 После получения адреса из внешнего источника необходимо привести такой адрес в «стандартный» для сервера вид. Для этого приложение использует возможности геокодирования. Геокодирование – это перевод произвольного адреса в координаты либо в адрес, сформированный по правилам сервиса поставщика геокодирования. Геокодирование бывает двух видов – прямое и обратное. В приложении используются возможности только прямого геокодирования [17].

3 Сформированные адреса сохраняются в файле на устройстве, где запущен сервер веб-приложения. Был выбран такой формат хранения данных, так как эта информация является «статичной» и зависит только от списка адресов.

4 Проверяется наличие информации об адресе и его показателях воды в базе данных. Важно отметить, что «хорошими» записями в Базе данных считаются записи, у которых есть параметры воды и которые обновлялись в текущем месяце. Все остальные записи считаются отсутствующими в базе данных и информацию о таких адресах необходимо обновить.

5 Для не отфильтрованных геокодированных адресов и их координат необходимо получить параметры воды. Для этого устанавливается соединение с сайтом *minskvodokanal.by* посредством использования *web-*драйвера через библиотеку *Selenium* на запущенном сервере. На сайте Минского водоканала можно просмотреть информацию о показателях воды по адресу в табличном формате, однако сайт не дает никакой визуальной оценки качества воды.

6 Полученная информация сохраняется в базу данных с отметкой времени получения показателей воды. На основе такой метки, как можно заметить выше, делаются различные фильтрации для экономии запросов во внешние источники.

На рисунке 2.2 приведена блок-схема алгоритма получения и сохранения информации о качестве воды для адреса.

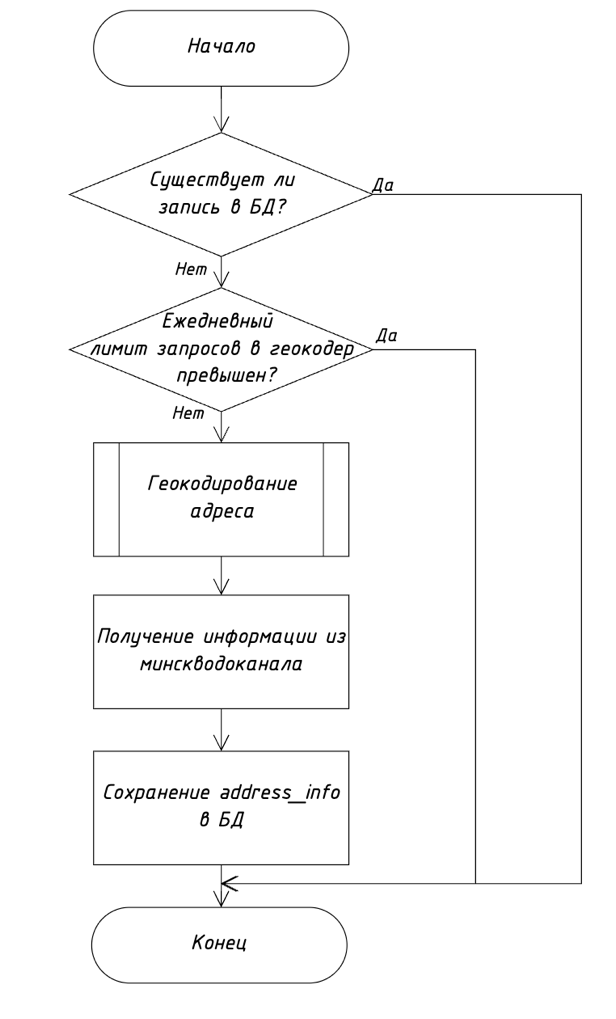


Рисунок 2.2 – Блок-схема алгоритма получения и сохранения информации о качестве воды для адреса

После сохранения информации об адресах жилых домов, необходимо вычислить средние показатели воды для гексагонов.

Алгоритм работы сервера для расчета средних показателей воды:

1 Извлекаются вся информация об адресах, сохраненная в базу данных. Так как в базе данных хранится информация не только о самых актуальных параметрах воды, но еще и историческая, то на этом этапе необходимо отфильтровать для каждого гексагона только последнюю запись по created\_at полю. Получившиеся записи являются основной для расчетов показателей воды гексагонов.

2 Сбор доступных резолюций для выполнения расчета. Для этого необходимо извлечь информацию из соответствующего конфига веб-приложения.

3 Итерируясь по всем адресам преобразовываем координаты в гексагоны соответствующей резолюции. Группируем показатели воды по получившемуся после преобразования *hex\_id.*

4 Вычисляем среднее значение для сгруппированных по hex\_id показателях воды.

5 Нормируем получившиеся величины. Для нормирования делим фактическое значение на предельно допустимое значение.

6 Вычисляем максимальное отклонение нормированных величин. Максимальное отклонение будем принимать за основной параметр для формирования цвета.

1. Переводим отклонение в цвет. Для этого используем цвет в формате *HSV*, который получаем, учитывая отклонение, которое находится в пределах от 0.0 до 1.0. После получения цвета в формате *HSV* преобразуем его в формат *RGB* при помощи стандартной библиотеки *colorsys*. Подробнее этот алгоритм перевода нормированных значений в цвет будет описан далее.
2. Сохраняем все информацию в базу данных. При этом сохраняем отметку времени расчета средних показателей. Такая отметка будет затем использована на фронтенде для отображения показателей воды по конкретному периоду времени.

На рисунке 2.3 приведена блок-схема алгоритма расчета средних показателей воды для гексагона.

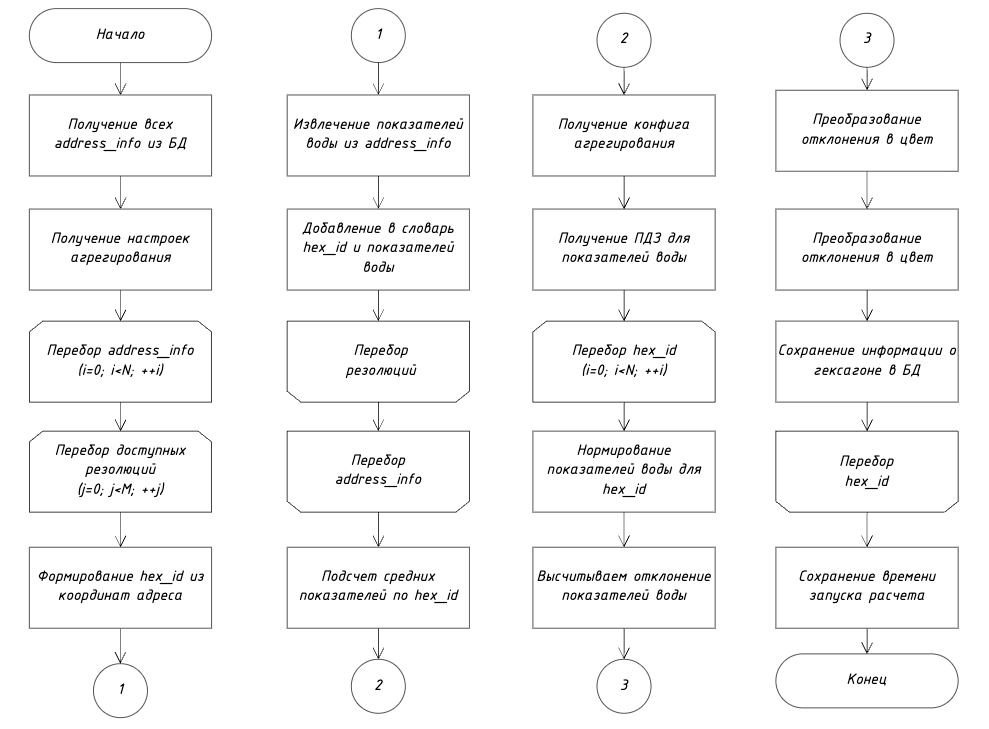


Рисунок 2.3 – Блок-схема алгоритма расчета средних показателей воды для гексагона

Для наилучшего описания алгоритма работы сервера была выбрана диаграмма взаимодействия, которая представлена на рисунке 2.4. Данная диаграмма используется, чтобы продемонстрировать последовательность взаимодействия компонент системы. Диаграмма отображает взаимосвязь между периодическими задачами, базой данных и внешними источниками (такими как *ato.by* и *minskvodokanal.by*).

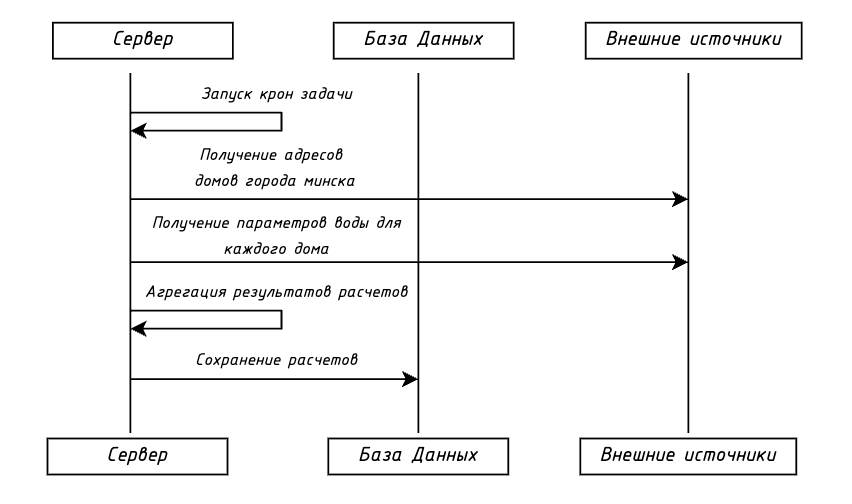


Рисунок 2.4 – Диаграмма взаимодействия при расчете параметров воды всех гексагонов города Минска

Описанные выше алгоритмы расчетов, взаимодействия и запуска сервера являются только частью системы. Основное время, которое пользователь проводит, используя веб-приложение, связано с фронтендом (клиентской частью). В интерфейсе веб-приложения поддержаны следующие возможности:

1 Отображение тепловой карты качества воды с помощью гексагональной сетки седьмой и восьмой резолюций.

2 Поиск по *hex\_id*.

3 Выбор гексагона на карте и просмотр его параметров.

4 Ввод адреса и просмотр показателей воды. Тут стоит отметить, что в таком варианте использования отличий между сайтом водоканала и веб-приложения нет. Оба веб-приложения предоставляют информацию в табличном виде, если запрос был сделан для конкретного адреса.

5 Просмотр параметров воды за определенный промежуток времени.

На рисунке 2.5 изображена главная страница веб-приложения, на которой можно увидеть все перечисленные функциональные особенности.

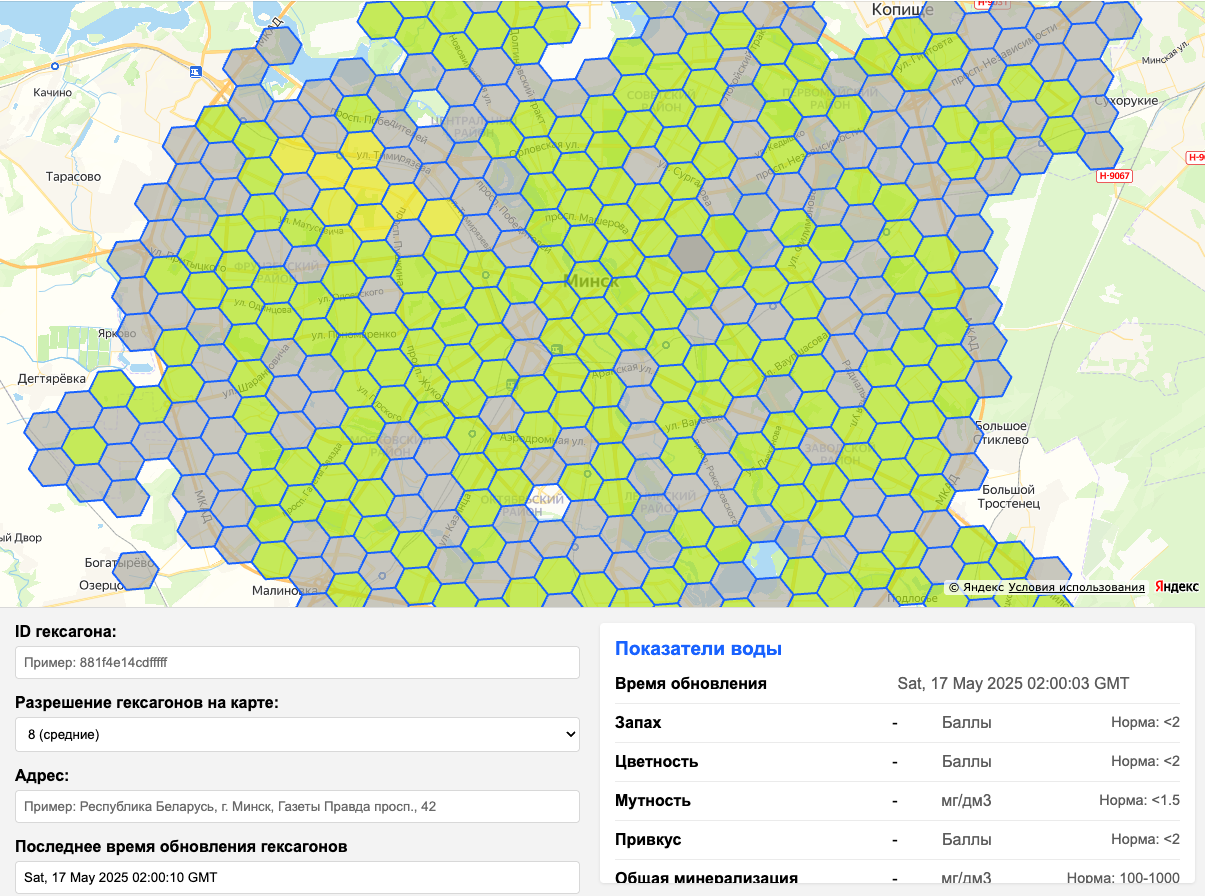


Рисунок 2.5 – Главная страница веб-приложения

После попадания на главную страницу пользователь может взаимодействовать с картой, перемещаться по ней в разных направлениях, приближать и удалять ее. Однако главное, что может делать пользователь – это получать информацию по показателях воды.

Алгоритм действия пользователя для получения показателей воды по гексагону:

1 Пользователь заходит на главную страницу веб-приложения.

2 Используя компьютерную мышь пользователь выбирает один из цветных гексагонов на карте. На карте есть места как цветных гексагонов, так и серых. Серый цвет обусловлен тем, что сервер веб-приложения еще не рассчитал параметры воды для адресов, которые находятся внутри этого гексагона. Это может произойти, если мы недавно запустили выполнение асинхронных задач или у нас закончилась квота для запросов в Минский водоканал. Пример подсветки гексагона при наведении мышью представлен на рисунке 2.6.

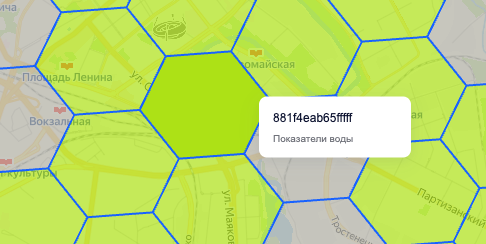


Рисунок 2.6 – Выделение гексагона при наведении мышью

3 После клика на выделенный гексагон, карта приближается к выбранному гексагону и на ней появляется отметка о местоположении. Местоположение отметки выбирается как центр гексагона, на который кликнул пользователь. Также в правом углу экрана появляется заполненная информация о показателях воды в этом гексагоне. Пример экрана при выборе гексагона приведен на рисунке 2.7.

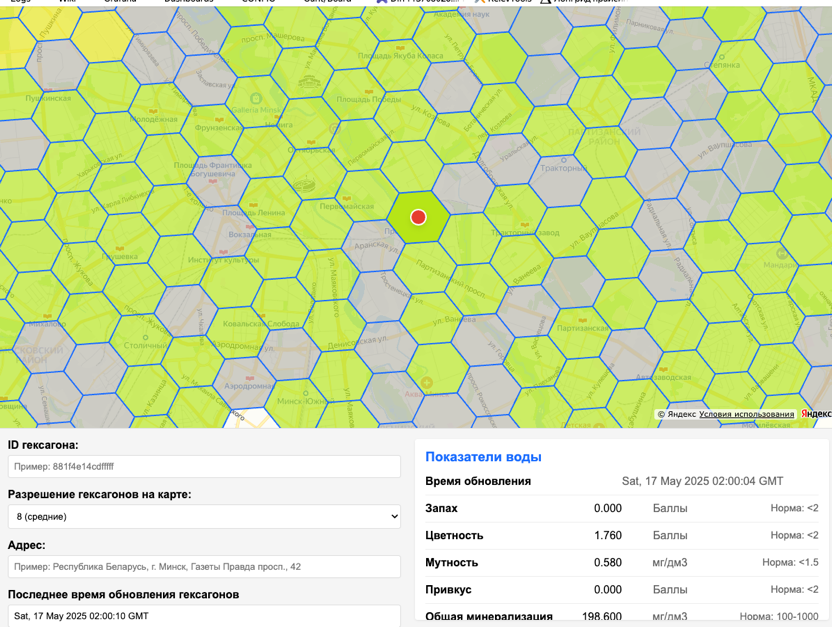


Рисунок 2.7 – Отображение показателей воды при клике по гексагону

Одним из вариантов использования приложения пользователем является просмотр показателей воды по конкретному адресу. Такой вариант использования уже существует и поддержан на сайте Минского водоканала, поэтому аналогичная вещь есть и в разработанном веб-приложении. Однако, одно из преимуществ подхода – это переиспользование данных и экономия запросов в геокодер. Существенный недостаток, который есть на сайте Минского водоканала – это запрос геокодирования на каждый введенный символ адреса. Разработанное веб-приложение имеет другой подход – полностью введенный адрес должен быть подтвержден при помощи клавиши *Enter.*

Алгоритм получения информации по введенному пользователем адресом:

1 Пользователь заходит на главную страницу веб-приложения.

2 В разделе селекторов (левый нижний угол) необходимо указать адрес интересующего нас дома. На рисунке 2.8 представлена форма для заполнения адреса в модуле селекторов для получения показателей воды.

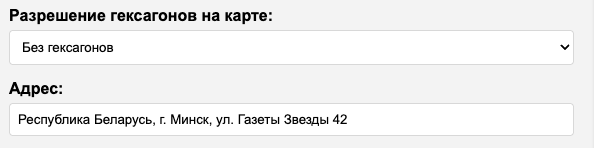


Рисунок 2.8 – Форма для заполнения адреса для получения показателей воды

3 Запрос с введенным адресом отправляется на сервер, где он обрабатывается по следующей логике: принимаем запрос, пытаемся найти адрес в базе данных, если адрес не был найден, геокодируем (приводим адрес в стандартным для базы данных вид) и пытаемся найти информацию еще раз. Если попытки найти в базе данных информацию об адресе оказались неудачными, то необходимо сделать дополнительное получение данных из сайта Минского водоканала.

4 Полученный из сервера ответ необходимо обработать и сформировать данные для области отображения показателей воды. На этом этапе происходит проставление предельно допустимых значений и единиц измерения, которые были получены из ответа сервера, в поля таблицы. Формируется список показателей воды, которые доступны по этому адресу. На сайте Минского водоканала этот список может отличаться в разных районах города.

Итоговая диаграмма взаимодействия для получения параметров воды по адресу представлена на рисунке 2.9.

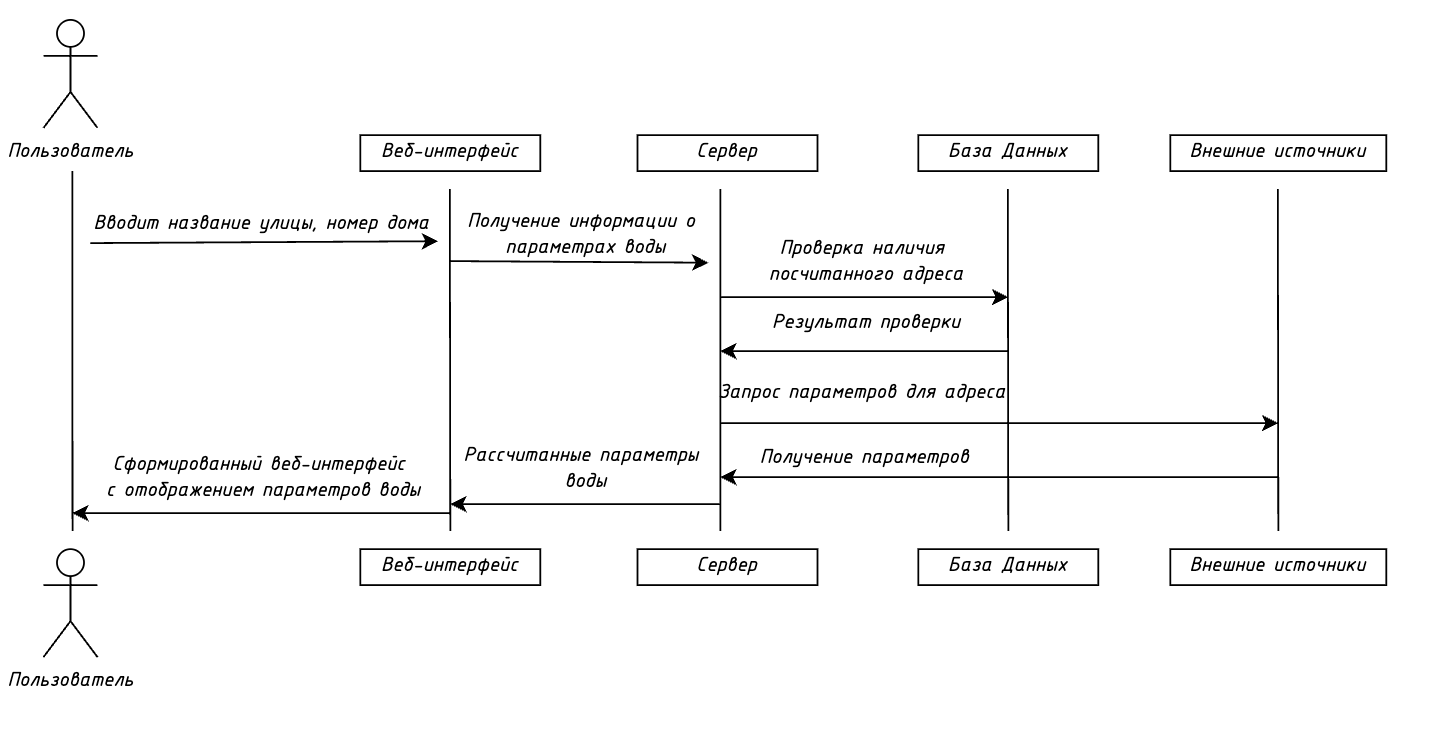


Рисунок 2.9 – Диаграмма взаимодействия пользователя с веб-приложением для получения показателей воды по адресу

Также на стороне сервера присутствует алгоритм декодирования адреса в необходимые для веб-приложения сущности: гексагон (*hex\_id*) и координаты. Алгоритм:

1 Полученный адрес с помощью *API* карт преобразуется в координаты, которые содержат широту и долготу.

2 Из интерфейса веб-приложения извлекается информация о текущей резолюции гексагонов. Резолюция гексагона – коэффициент размера области, которую покрывает гексагон. Формирование гексагонов происходит посредством библиотеки *h3*, которую разработал *Uber*.

3 Формируется сущность *Address*, которая далее используется в системе как единый интерфейс обращения с координатами, названием адреса в привычном для человека формате и *hex\_id*.

Алгоритм формирования цвета гексагона был выбран следующий:

1 Нормализуем значения параметров воды согласно ПДК.

2 После нормализации находим максимальное отклонение.

3 Преобразуем отклонение в цвет: для этого используем формат *HSV* (*hue*, *saturation*, *value*), где первый параметр – это цвет, *saturation* – насыщенность и *value* – яркость. Для того чтобы преобразовать отклонение в цвет необходимо задать два граничных значения – это красный и зеленый цвета, а также зафиксировать яркость и насыщенность [18].

4 Преобразуем *HSV* в *RGB*. Для этого используем одну из общих библиотек, которые поставляются вместе с языком программирования *Python, colorsys*. [19].

Таким образом алгоритм работы программы отображает основные этапы спроектированной системы. Система может как использовать данные, посчитанные с помощью асинхронного процесса, так и запрашивать данные с помощью точечного адреса. Также в этом разделе был отражен алгоритм перевода адресов в гексагоны и координаты и продемонстрирован алгоритм конвертации параметров воды в цветовую схему.

## 2.3 Структура базы данных

Для хранения и обработки информации в рамках программного средства используется реляционная база данных, разработанная с учетом требований к целостности, производительности и масштабируемости. Структура базы данных основана на концептуальной модели предметной области и реализована с использованием системы управления базами данных (СУБД) ***PostgreSQL* [20].**

База данных веб-приложения для усовершенствования мониторинга качества воды состоит из таблиц *address\_info*, в которой хранится информация о результатах геокодирования и извлечения информации из сайта Минского водоканала, *hexagons*, в которой хранится агрегированная информация о качестве воды в гексагонах, *m2m\_hexagons\_address*, которая является промежуточной таблицей для реализации связи многие ко многим.

Таблица hexagons имеет следующую структуру: поле *hex\_id*, которое является главным для поиска и которое является идентификатором гексагона, *created\_at* – *timestamp*, который отражает момент попадания гексагона в систему для расчета его параметров воды, *hex\_res* – резолюция сегмента, которая добавлена для более простого взаимодействия селекторов с таблицей, а также *hex\_color* и *avg\_water\_parameters*. Поле *created\_at* необходимо, чтобы фильтровать данные, которые оказались либо слишком старыми, либо слишком новыми для промежутка времени, выбранного в селекторах. Поле *hex\_color* – это строка, которая отвечает за то, какой цвет сейчас должен быть у гексагона в формате *rgb*. Поле *avg\_water\_parameters* содержит все средние показатели воды для адресов, которые попадают в текущий гексагон [21].

Данная сущность и ее атрибуты представлены в таблице 2.1.

Таблица 2.1 – Отношение *hexagons*

|  |  |  |  |
| --- | --- | --- | --- |
| Атрибут | Тип данных | Ограничения | Описание |
| *id* | *bigint* | *PRIMARY KEY* | Уникальный идентификатор записи о гексагоне |
| *created\_at* | *timestamp* | *DEFAULT now()*  *NOT NULL* | Дата и время создания записи |
| *hex\_id* | *text* | *NOT NULL* | ID гексагона |
| *hex\_res* | *integer* | *NOT NULL* | Резолюция гексагона |
| *hex\_color* | *text* | *NOT NULL* | Цвет гексагона. Представлен в формате r, g, b |
| *avg\_water\_parameters* | *json* |  | Средний показатели воды для гексагона. Хранятся в json формате |

Таблица *address\_info* содержит информацию о точечных адресах. Такая таблица полезна для двух вещей: экономим (переиспользуем) уже известную информацию о геокодировании и храним показатели воды по каждому из домов в базе данных. Таблица имеет следующую схему: *created*\_*at*, *address*, *latitude*, *longitude*, *water*\_*parameters*. Поле *created*\_*at*, как и для таблицы *hexagons*, является timestamp’ом добавления информации в базу данных. Поле *address*, *latitude*, *longitude* представляют собой информацию о адресе, которая была получена после успешного геокодирования. Такая операция необходима для того, чтобы в базе данных не хранились дубликаты адресов. Поле *water*\_*parameters*, которое содержит показатели воды по конкретному дому. Данная сущность и ее атрибуты представлены в таблице 2.2.

Таблица 2.2 – Отношение *address*\_*info*

|  |  |  |  |
| --- | --- | --- | --- |
| Атрибут | Тип данных | Ограничения | Описание |
| *id* | *bigint* | *PRIMARY KEY* | Уникальный идентификатор записи об адресе |
| *created\_at* | *timestamp* | *DEFAULT now()*  *NOT NULL* | Дата и время создания записи |
| *address* | *Text* | *NOT NULL* | Полный адрес для дома. Такой адрес получается при помощи геокодирования |
| *latitude* | *numeric* | *NOT NULL* | Координаты (широта) адреса на карте. Координаты получается при помощи геокодирования |
| *longitude* | *numeric* | *NOT NULL* | Координаты (долгота) адреса на карте. Координаты получается при помощи геокодирования |
| *water\_parameters* | *json* |  | Показатели воды для адреса. Хранятся в json формате |

Таблица *m2m\_hexagons\_address* используются для реализации связи многие ко многим. Такая связь полезна для двух вещей: получить все адреса, которые относятся к гексагону, и получить все гексагоны, которые относятся к адресу.

Таким образом, база данных системы обеспечивает гибкость и масштабируемость, позволяя эффективно хранить и обрабатывать данные о гексагонах, времени их создания, а также параметрах воды. Такая структура позволяет системе анализировать средние показатели и выбирать временные промежутки для отображения.

## 2.4 Выводы и оценка результатов разработки

Разработка веб-приложения для визуализации мониторинга качества воды успешно завершена. Система полностью соответствует поставленным требованиям и обеспечивает эффективное решение задач сбора, анализа и наглядного представления данных о состоянии водных ресурсов.

Ключевым преимуществом приложения является его специализированный функционал, ориентированный на визуализацию качества воды через гексагональную сетку. Система автоматически загружает и анализирует данные с сайта Минскводоканала, преобразуя табличные показатели в интуитивно понятную цветовую карту. Пользователи могут изучать детализированную информацию по районам, применять фильтры для анализа конкретных параметров воды и сравнивать данные за разные периоды.

В процессе разработки успешно решены задачи интеграции с внешними источниками данных, проектирования алгоритмов классификации показателей воды и реализации интерактивной карты на основе *JS API* Яндекс Карт. Система прошла тестирование на реальных данных, подтвердив точность визуализации и стабильность работы.

База данных системы, построенная на *PostgreSQL*, обеспечивает надежное хранение и обработку данных. Структура базы данных включает таблицы для хранения информации об адресах города Минска и для хранения агрегированных (средних) показателей воды для гексагонов. Такая структура позволяет системе эффективно анализировать данные и предоставлять потребителям актуальную информацию.

В процессе разработки успешно решены задачи проектирования архитектуры системы, реализации бизнес-логики, интеграции с картографическими сервисами. Система прошла тестирование на реальных данных, подтвердив свою работоспособность и соответствие требованиям.

Таким образом, разработанное веб-приложение представляет собой эффективный инструмент для мониторинга качества воды, сочетающий удобство использования с мощной аналитической составляющей.

# 3 АНАЛИЗ НАДЕЖНОСТИ ВЕБ-ПРИЛОЖЕНИЯ

## 3.1 Расчет надежности по модели сложности

Модель сложности программного обеспечения основывается на предположении, что уровень безошибочности продукта может быть предсказан с использованием метрик сложности. Это справедливо для непреднамеренных уязвимостей, так как, чем сложнее и больше программа, тем выше вероятность ошибок при её написании и модификации. Для расчета надежности программного средства по модели сложности используются метрики размера, сложности потока управления, сложности потока данных, а также объектно-ориентированные метрики.

Модель сложности программного обеспечения предполагает, что уровень безотказной работы проекта может быть определен с помощью набора параметров, называемых метриками сложности. В частности этот метод особенно эффективен при работе с непреднамеренными ошибками и уязвимостями, так как комплексность программы предполагает повышенный шанс совершения ошибок при проектировании и создании веб-приложения.

Для расчета надежности веб-приложения по усовершенствованию мониторинга качества воды в городе Минске по модели сложности использовались метрики размера, метрики сложности потока управления, метрики сложности потока данных, объектно-ориентированные метрики.

Исходные данные для модели сложности программного средства приведены в таблице 3.1.

Таблица 3.1 – Исходные данные разработанной системы

|  |  |
| --- | --- |
| *n*1 | 15 |
| *n*2 | 30 |
| *N*1 | 170 |
| *N*2 | 85 |
| *n\** | 136 |
| *P* | 5 |
| *M* | 75 |
| *C* | 5 |
| *T* | 0 |

К метрикам размера относят объём (*V*) и потенциальный объём (*V\**) программного средства.

Объём программного средства *V* рассчитывают по формуле (3.1):

|  |  |
| --- | --- |
|  | (3.1) |

|  |  |  |  |
| --- | --- | --- | --- |
| где | *n*1 | – | число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов); |
|  | *n*2 | – | число уникальных операндов программы (словарь операндов); |
|  | *N*1 | – | общее число операторов в программе; |
|  | *N*2 | – | общее число операндов в программе. |

Потенциальный объем программы определяется по формуле (3.2):

|  |  |
| --- | --- |
|  | (3.2) |

|  |  |  |  |
| --- | --- | --- | --- |
| где |  | – | теоретический словарь программы. Теоретический словарь программы равен сумме количества операторов и операндов программы, при условии того, что нужная функциональность уже реализована в программе [22]. |

Для разрабатываемого программного средства значения объема и потенциального объема равны:

.

Для определения метрик сложности потока управления используются следующие характеристики:

– CL – абсолютная сложность программы, характеризующаяся количеством операторов условия;

– cl – относительная сложность программы, определяемая как отношение CL к общему числу операторов;

– CLI – максимальный уровень вложенности операторов условия.

Для разрабатываемого программного средства значения этих характеристик равны: CL =178, cl = 79, CLI = 57.

Метрика сложности потока данных, предложенная Чепиным, оценивает информационную прочность программного модуля на основе анализа использования переменных из списка ввода-вывода. Все переменные делятся на четыре группы:

– *Р* – вводимые переменные для расчетов и обеспечения вывода. Примером может служить используемая в программах лексического анализатора переменная, содержащая строку исходного текста программы, т. е. сама переменная не модифицируется, а только содержит исходную информацию;

*– М* – модифицируемые, или создаваемые внутри программы переменные;

*– С* – переменные, участвующие в управлении работой программного модуля (управляющие переменные);

*– Т* – не используемые в программе («паразитные») переменные.

Для разрабатываемого веб-приложения значения *P*, *M*, *C*, *T* равны 5, 75, 5, 0 соответственно. Метрика Чепина рассчитывается по формуле (3.3):

(3.3)

Для разрабатываемого веб-приложения значение метрики Чепина равно:

Следующие характеристики являются реализацией метрики Холстеда. Для измерения теоретической длины программы используется формула (3.4):

(3.4)

Для разрабатываемого программного средства = 190,81.

Другой характеристикой, принадлежащей метрикам стилистики и понятности, является уровень качества программирования *L*, определяемый формулой (3.5):

(3.5)

Исходным для введения этой характеристики является предположение о том, что при снижении стилистического качества программирования уменьшается содержательная нагрузка на каждый компонент программы и, как следствие, расширяется объем реализации исходного алгоритма. Учитывая это, можно оценить качество программирования на основании степени расширения текста относительно потенциального объема *V\**. Очевидно, для идеальной программы *L* = 1, а для реальной всегда *L* < 1.

Подставив необходимые значения в формулу (3.5), получаем, что для разрабатываемого веб-приложения *L* = 0,69.

Аппроксимированный уровень качества программирования L^ рассчитывается по формуле (3.6):

(3.6)

Для разрабатываемого веб-приложения = 0,05.

Интеллектуальные усилия на разработку программы E рассчитываются по формуле (3.7):

Для разрабатываемого веб-приложения *E* = 2034,64

(3.7)

Для оценки объектно-ориентированных характеристик программного средства используются метрики Чидамбера и Кемерера:

– *WMC* – суммарная сложность всех методов класса. Подсчитывается количество методов в классе, а также сложность каждого метода (например, цикломатическая сложность), затем суммируется показатель сложности всех методов в классе. Численный показатель сложности должен быть разумно низким. Можно использовать упрощенный способ расчета, в этом случае подсчитывается только количество методов в классе, чем меньше методов содержит класс, тем выше его оценка качества;

– *DIT* – глубина дерева наследования. Значением метрики для класса будет длина максимального пути от данного класса до корневого класса в иерархии классов. С одной стороны, большое значение этой метрики указывает на высокую сложность проекта (в связи с использованием большого количества классов и методов), с другой стороны, подразумевает, что многие методы могут использоваться многократно, что является хорошим показателем;

– *NOC* – количество потомков. Значением этой метрики будет количество непосредственных классов-наследников рассматриваемого класса. С увеличением количества наследников возрастает многократность использования методов и свойств классов, что является хорошей характеристикой, но при этом возникает риск неправильного использования некоторых классовнаследников. Поэтому значение данной метрики должно быть разумно низким;

– *CBO* – сцепление между классами. Рассчитывается количество классов, с которыми соединен рассматриваемый класс. Соединение образуется в результате вызова рассматриваемым классом метода или свойства другого класса;

– *RFC* – отклик для класса. При расчете показателя для класса используется количество собственных методов класса, а также количество методов, которые могут быть вызваны из рассматриваемого класса. Большое значение этой метрики свидетельствует о высокой сложности класса, что негативно сказывается на процессе тестирования и отладки;

– *LCOM* – недостаток сцепления методов. Значение метрики показывает, насколько в классе количество пар методов, не связанных по свойствам класса, преобладает над количеством пар методов, связанных по свойствам класса.

Для разрабатываемого веб-приложения значения этих метрик равны: *WMC* = 201, *DIT* = 5, *NOC* = 3, *CBO* = 13, *RFC* = 64, *LCOM* = 7.

Исходные данные для метрики сложности представлены в таблице 3.2.

Таблица 3.2 – Исходные метрики для модели сложности

|  |  |  |  |
| --- | --- | --- | --- |
| Метрики ПС | amin | amax | ai |
| *V* | 108 | 1364 | 1400,4 |
| *V\** | 540 | 83362,41 | 963,9 |
| *CL* | 7 | 368 | 178 |
| *cl* | 25 | 1278 | 79 |
| *CLI* | 41 | 4213 | 57 |
| *Q* | 67 | 3589 | 170 |
| *N^* | 6 | 186 | 190,81 |
| *L* | 0,0054 | 2 | 0,69 |
| *L^* | 0 | 3 | 0,05 |
| *E* | 29 | 967 | 2034,64 |
| *WMC* | 13 | 299 | 201 |
| *DIT* | 1 | 8 | 5 |
| *NOC* | 1 | 32 | 3 |
| *CBO* | 1 | 27 | 13 |
| *RFC* | 1 | 163 | 64 |
| *LCOM* | –39 | 387 | 7 |

Исходя из полученных метрик рассчитаны нормативный и фактический уровни каждой метрики (формулы (3.8) и (3.9) соответственно):

(3.8)

(3.9)

По полученным результатам определен дискриминант каждой метрики, используя формулу (3.10):

(3.10)

Результаты вычислений по формулам (3.8), (3.9) и (3.10) представлены в таблице 3.3.

Таблица 3.3 – Результаты вычисления

|  |  |  |  |
| --- | --- | --- | --- |
| Метрики ПС | xmin | xфi | di |
| V | 0,07917888563 | 1,026686217 | -0.00223638 |
| V\* | 0,006477739787 | 0,01156276552 | 0.557359718 |
| CL | 0,01902173913 | 0,05434782609 | 0.020697812 |
| cl | 0,01956181534 | 0,02034428795 | 0.302817542 |
| CLI | 0,009731782578 | 0,009969143128 | 0.716539671 |
| Q | 0,01866815269 | 0,04736695458 | 0.382591776 |
| N^ | 0,03225806452 | 1,102150538 | -0.00084029 |
| L | 0,0027 | 0,345 | 0.005159476 |
| L^ | 0 | 0,01666666667 | 0 |
| E | 0,02998965874 | 2,105305067 | -0.01622308 |
| WMC | 0,04347826087 | 0,6722408027 | 0.022161918 |
| DIT | 0,125 | 0,25 | 0.085714286 |
| NOC | 0,03125 | 0,09375 | 0.311827957 |
| CBO | 0,03703703704 | 0,2592592593 | 0.041420118 |
| RFC | 0,006134969325 | 0,1226993865 | 0.009548611 |
| LCOM | -0,1007751938 | 0,05167958656 | -4.96981891 |

Риск снижения надежности рассчитывается по формуле (3.11):

(3.11)

|  |  |  |  |
| --- | --- | --- | --- |
| где | *λi* | – | весовые коэффициенты для каждой метрики. Для упрощения расчетов можно считать, что все метрики вносят равный вклад в надежность, т.е. λi = 1/16. |

Для разрабатываемого веб-приложения вероятность безотказной работы рассчитывается по формуле (3.12):

(3.12)

Подставив в формулу (3.12) значения *R* равное 0,093 получаем, что вероятность безотказной работы веб-приложения по модели сложности составляет 0,907.

## 3**.2 Расчет надежности по модели Джелинского – Моранды**

Модель Джелинского – Моранды используется для оценки надежности программного обеспечения на основе данных о времени между обнаружением ошибок. Такая модель строится на основе следующих допущений [23]:

1 Интенсивность обнаружения ошибок λ(t) пропорциональна текущему числу ошибок в программном средстве, т. е. числу оставшихся ошибок.

2 Все ошибки одинаково вероятны, и их появления независимы.

3 Каждая ошибка имеет один и тот же порядок серьезности.

4 Время до следующего отказа (ошибки) распределено экспоненциально.

5 Программное средство функционирует в среде, близкой к реальной.

6 Ошибки постоянно корректируются без внесения в программное средство новых.

7 λ(t) = const в интервале между двумя соседними ошибками.

Метод оценивает вероятность безотказной работы веб-приложения и использует формулы (3.13) и (3.14) для вычисления этого значения:

|  |  |
| --- | --- |
|  | (3.13) |

|  |  |
| --- | --- |
|  | (3.14) |

В данном разделе проведем расчет надежности программного средства на основе модели Джелинского – Моранды. Общее время тестирования программы составило 17 дней. За это время было обнаружено 16 ошибок. Время между обнаружением ошибок (Xi) приведено в таблице 3.4.

Таблица 3.4 – Время между ошибками (Xi) в часах:

|  |  |
| --- | --- |
| Номер ошибки (i) | Время между ошибками (Xi) |
| 1 | 5 |
| 2 | 8 |
| 3 | 10 |
| 4 | 12 |
| 5 | 14 |
| 6 | 7 |
| 7 | 9 |
| 8 | 16 |
| 9 | 17 |
| 10 | 18 |
| 11 | 19 |
| 12 | 20 |
| 13 | 21 |
| 14 | 22 |
| 15 | 23 |
| 16 | 25 |

Для расчёта коэффициента *B* необходимо найти решения уравнения (3.15):

|  |  |
| --- | --- |
|  | (3.15) |

Константа *A* – величина, зависящая от данных и равная значению, которое подходит под условие (3.16):

|  |  |
| --- | --- |
|  | (3.16) |

Для выполнения расчета примем *A* = 9.

Для нахождения *B* выполним замену (заменим *B* + 1 на *m*) и найдем минимально возможную разность между и . Также на *m* накладывается условие, что оно должно быть больше, чем *n* + 1. Такое число отражает число прогнозируемых (пока не обнаруженных) ошибок.

Функции *f*(*m*) и *g*(*m*, *A*) определяются следующим образом по формулам (3.17) и (3.18):

|  |  |
| --- | --- |
|  | (3.17) |

|  |  |
| --- | --- |
|  | (3.18) |

В ходе расчетов для различных значений *m* (диапазон значений был выбран от 17 до 80) получаем, что минимальная разность достигается при *m*=51, что соответствует общему числу ошибок *B*=50 (так как *B*=*m*−1).

Неизвестный коэффициент *K* рассчитывается по формуле 3.19:

|  |  |
| --- | --- |
|  | (3.19) |

Подставляя найденное значение в формулу, получаем:

|  |
| --- |
|  |

Интенсивность отказов λ(t)после обнаружения 16 ошибок рассчитывается по формуле (3.13) и равно:

|  |
| --- |
|  |

Вероятность безотказной работы *P*(*t*) на интервале *t*=1 час определяется по формуле (3.14) и равно:

|  |
| --- |
|  |

Среднее время до появления следующей ошибки (*i* + 1) рассчитывается по формуле (3.20):

|  |  |
| --- | --- |
|  | (3.20) |

Таким образом получаем, что время до появления следующей ошибки равно приблизительно 18 часов.

Время до окончания тестирования оценивается как сумма времени до обнаружения оставшихся ошибок. Такое время рассчитывается по формуле (3.21):

|  |  |
| --- | --- |
|  | (3.21) |

Предположив, что ошибки будут находиться через среднее значение времени, получим = 108 дней.

Таким образом, модель Джелинского – Моранды позволяет оценить надежность программного средства и спрогнозировать время, необходимое для обнаружения и исправления всех ошибок.

## 3.3 Расчет надежности по модели Муса

Проведем расчет надежности по модели Муса. В этой модели надежность программного средства на этапе эксплуатации оценивается по результатам тестирования. Пусть Т – суммарное время тестирования, М – число отказов, произошедших за время тестирования. Тогда по модели Муса средняя наработка до отказа после тестирования определяется по формуле (3.22).

|  |  |
| --- | --- |
|  | (3.22) |

В этой формуле τ0 – средняя наработка до отказа до начала тестирования, С – коэффициент, учитывающий уплотнение тестового времени по сравнению с временем реальной эксплуатации. Неизвестный параметр τ0 можно оценить из соотношения (3.23).

|  |  |
| --- | --- |
|  | (3.23) |

В этой формуле *N* – первоначальное число ошибок в программной средстве, которое можно оценить с помощью другой модели, позволяющей определить *N* на основе статистических данных, полученных при тестировании. *К* – коэффициент проявления ошибок, который определяется эмпирическим путем по однотипным программам, обычно это значение изменяется от 1,5·10–7 до 4·10–7. *f* – средняя скорость исполнения программы, деленная на число команд [24].

Средняя скорость исполнения одного оператора программного средства определяется по формуле (3.24).

|  |  |
| --- | --- |
|  | (3.24) |

Длительности этапов тестирования составляют t1 = 50 ч, t2 = 42 ч, t3 = 36 ч. Число отказов на первом этапе m1 =8, на втором – m2 = 5, на третьем – m3 = 3. Средняя скорость исполнения программного средства = 107 операторов/ч, количество операторов в программном средстве = 968. Исходные данные для модели Муса приведены в таблице 3.5.

Таблица 3.5 – Исходные данные для модели Муса

|  |  |
| --- | --- |
| *C* | 12 |
| *N* | 16 |
| *K* | 3,0210-7 |
| *A* | 107 |
| *B* | 968 |
| *t*1 | 50 |
| *t*2 | 42 |
| *t*3 | 36 |
| *m*1 | 8 |
| *m*2 | 5 |
| *m*3 | 3 |
| *t* | 128 |

Определим надежность системы для периода эксплуатации t = 128 ч.

Найдем среднюю скорость выполнения одного оператора по формуле (3.24):

Первоначальное количество ошибок в программном средстве N равно 16. Найдем формуле (3.23).

Средняя наработка до отказа после тестирования на этапе эксплуатации программного обеспечения рассчитывается по формуле (3.22).

Найдем надежность программного средства для периода эксплуатации t, равному 128 часам, по формуле (3.25).

|  |  |
| --- | --- |
|  | (3.25) |

Надежность, рассчитанная по формуле (3.25), равна:

Таким образов в этом разделе были посчитаны метрики надежности веб-приложения для усовершенствованной визуализации мониторинга качества воды с использованием цветовых гексагонов на карте города Минска. Метрики надежности веб-приложения были рассчитаны по модели сложности, по модели Джелинского-Моранда, по модели Муса и составили 0,907, 0,95, 0,87 соответственно. Результаты расчета надежности программного средства представлены на плакате ГУИР.110101.006 ПЛ.

# 4 ТЕСТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ ПО УСОВЕРШЕНСТВОВАНИЮ ВИЗУАЛИЗАЦИИ МОНИТОРИНГА КАЧЕСТВА ВОДЫ ПОСРЕДСТВОМ ВВЕДЕНИЯ ЦВЕТОВЫХ ГЕКСАГОНОВ НА КАРТЕ ГОРОДА МИНСКА

Для функционального тестирования приложения были разработаны и использованы наборы *Test* *case* Таблица 4.1 [22].

Таблица 4.1 – Тест кейсы веб-приложения

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Требование | Модуль/Подмодуль | Описание тест-кейса | Ожидаемый результат | Статус |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | Приложение выводит на экран карту при входе на главную страницу | Карта / *JS API* Яндекс Карт | 1. Запустить веб-приложение 2. Убедиться, что присутствует карта | 1. Приложение запускается без ошибок 2. На экране присутствует карта 3. По карте можно перемещаться: двигать, приближать и отдалять. | Соответствует ожидаемому |
| 2 | Приложение выводит на экран гексагоны на карте при входе в веб-приложение | Карта / Слой пользовательских объектов | 1. Запустить веб-приложение 2. Убедиться, что установлено соединение с сервером (в консоли нет ошибок) 3. Убедиться, что на карте отображаются гексагоны с резолюцией 8. | 1. Приложение запускается без ошибок 2. На экране присутствует карта 3. Площадь Минска покрыта гексагональной стекой. Цвет клеток может быть любой. | Соответствует ожидаемому |

Продолжение таблицы 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 3 | При наведении на гексагон его цвет меняется на более темный | Карта / Слой пользовательских объектов | 1. Запустить веб-приложение 2. Убедиться, что карта присутствует 3. Убедиться, что на карте есть гексагональная сетка 4. Провести курсор поверх любого из гексагонов | 1 Приложение запускается без ошибок  2 На экране присутствует карта  3 При наведении на гексагон его цвет меняется на более темный. | Соответствует ожидаемому |
| 4 | При наведении на гексагон отображаются его показатели воды | Показатели воды | 1. Запустить веб-приложение 2. Убедиться, что карта присутствует 3. Убедиться, что на карте есть гексагональная сетка 4. Провести курсор поверх любого из гексагонов | 1 Приложение запускается без ошибок  2 На экране присутствует карта  3 При наведении на гексагон показатели заполняются значениями. | Соответствует ожидаемому |
| 5 | При наведении на гексагон выводиться на экран его *ID* справа от курсора | Карта / Слой подсказок в API Яндекс Карт | 1. Запустить веб-приложение 2. Убедиться, что карта присутствует 3. Убедиться, что на карте есть гексагональная сетка 4. Провести курсор поверх любого из гексагонов | 1 Приложение запускается без ошибок  2 На экране присутствует карта  3 При наведении на гексагон справа от гексагона появляется небольшое окно, где присутствует *ID* гексагона. | Соответствует ожидаемому |

Продолжение таблицы 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 6 | При нажатии на гексагон карта увеличивается и смещается к центру гексагона | Карта / *JS API* Яндекс Карт | 1. Убедиться, что на карте есть гексагональная сетка 2. Нажать левой кнопкой мыши на любой из гексагонов | 1. Смещение центра карты из центра Минска в центр гексагона 2. Увеличение приближения карты | Соответствует ожидаемому |
| 7 | При нажатии на гексагон фиксируются параметры воды | Карта / Показатели воды | 1 Убедиться, что на карте есть гексагональная сетка  2 Нажать левой кнопкой мыши на любой из гексагонов | 1. Отображение показателей воды гексагона 2. При наведении мышью на другие гексагоны параметры воды не сменяются | Соответствует ожидаемому |
| 8 | При нажатии на гексагон фиксируется его *hex*\_*id* в поле селектора | Селекторы / Группа селекторов для гексагонов | 1 Убедиться, что на карте есть гексагональная сетка  2 Нажать левой кнопкой мыши на любой из гексагонов | 1. В селекторах в поле для *hex*\_*id* появляется *id* гексагона | *D-1* |

Продолжение таблицы 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 9 | При нажатии на гексагон, с уже выбранным гексагоном, сменяется точка на карте | Карта / *JS API* Яндекс Карт | 1 На карте должен быть выбран гексагон   1. 2 Выбираем еще один гексагон на карте | 1 Изменение показателей воды   1. Перемещение карты в центр другого гексагона 2. Перемещение пина на новый гексагон | Соответствует ожидаемому |
| 10 | При нажатии *ESC* карта возвращается в изначальное состояние | Карта / *JS API* Яндекс Карт | 1. Выбрать любой гексагон 2. Убедиться, что центр карты сместился относительно предыдущего положения 3. Нажать *ESC* | 1. Центр карты возвращается к своему изначальному положению 2. Показатели воды загуляются 3. Отсутствует пин на карте. | Соответствует ожидаемому |
| 11 | При вводе некорректного *id* должно приходить уведомление | Селекторы / Группа селекторов для гексагонов | 1. Ввести некорректный *ID* для гексагона | 1. В верхней части экрана должно появится уведомление о некорректности формата *hex*\_*id* | *D-2* |

Продолжение таблицы 4.1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | | 2 | 3 | 4 | | 5 | 6 | |
| 12 | Пользователь может  менять резолюцию гексагонов | | Селекторы / Группа селекторов для гексагонов | 1. Открыть список доступных резолюций 2. Выбрать любую отличную от 8-ой | 1. С карты пропали все гексагоны с резолюцией восемь 2. На карте появились гексагоны с необходимой резолюцией | | Соответствует ожидаемому |
| 13 | Пользователь вводит дату и время периода, за который он собирается смотреть данные | | Селекторы / Группа селекторов для даты и времени | 1. Открыть вкладку с датой и временем 2. Выбрать необходимую дату и время | 1. На карте должна поменяться ситуация с расцветкой гексагонов 2. Кликнуть на любой гексагон 3. Значение в поле «Время обновления» должно находится в пределах выбранных в селекторах даты и времени. | | Соответствует ожидаемому |
| 14 | Ввод гексагона не поддерживаемой резолюцией | | Селекторы / Группа селекторов для гексагонов | 1. Открыть поле для ввода *hex\_id* 2. Ввести в поле *hex*\_*id* с 9-ой резолюцией | 1. Подсвечивание поля для ввода красным цветом. 2. Уведомление пользователя о слишком большом размере гексагона | | Соответствует ожидаемому |

Продолжение таблицы 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 15 | Пользователь вводит некорректный адрес для Минска | Селекторы / Группа селекторов для адресов | 1. Открыть поле для ввода адреса 2. Ввести некорректный адрес | 1. Поле для ввода адреса подсвечивается красной рамкой | Соответствует ожидаемому |
| 16 | Пользователь вводит корректный адрес для Минска | Селекторы / Группа селекторов для адресов | 1. Открыть поле для ввода адреса 2. Ввести корректный адрес | 1. На карте не отображаются гексагоны 2. Более сильное приближение камеры к адресу, чем у гексагонов 3. На карте стоит отметка адреса красной точкой 4. Показатели воды заполнены | Соответствует ожидаемому |
| 17 | Фиксация гексагона и поиск по адресу | Карта / Слой пользовательский объектов | 1. Убедиться, что гексагональная сетка отображается 2. Выбрать случайный гексагон 3. Ввести адрес | 1. На карте не отображаются гексагоны 2. На карте существует только одна красная точка, указывающая на введенный адрес 3. Показатели воды заполнены | Соответствует ожидаемому |

Продолжение таблицы 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 18 | Пользователь ввел один и тот же адрес два раза подряд | Селекторы / Группа селекторов для адресов | 1. Выбрать поле для ввода данных 2. Ввести корректный адрес | 1. На карте отобразиться информация о показателях воды и карта сместиться к точке 2. В Базе данных находится создалась новая запись (единственная) | Соответствует ожидаемому |
| 19 | Пользователь ввел адрес, затем ввел *hex*\_*id* | Селекторы / Общая группа селекторов адресов и гексагонов | 1. Вводим корректный адрес в селекторах 2. Вводим корректный *hex*\_*id* в селекторах | 1. Отображение информации об адресе 2. После введенного *hex*\_*id* удаление отметки адреса на карте 3. Построение гексагональной сетки. 4. Перемещение камеры в этот гексагон | Соответствует ожидаемому |
| 20 | Пользователь зафиксировал гексагон на карте, после чего сменил резолюцию | Карта / Слой пользовательских объектов | 1. Выбрать любой гексагон на карте (через селекторы или через интерфейс) 2. Сменить резолюцию гексагона через селекторы | 1. Отображение отметки на карте выбранного гексагона 2. Обновление карты после смены резолюции 3. Отсутствует метка зафиксированного гексагона | *D-3* |

По полученным результатам выполнения тест-кейсов был составлен отчет о выявленных дефектах, таблица 4.2.

Таблица 4.2 – Описание найденных дефектов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Название дефекта | Важность | Алгоритм воспроизведения | Фактический результат | Ожидаемый результат | Прило-жение |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| *D*-1 | При нажатии на гексагон фиксируется его *hex*\_*id* в поле селектора | Сред­няя | 1 Убедиться, что на карте есть гексагональная сетка  2 Нажать левой кнопкой мыши на любой из гексагонов | Нет выбранного *hex\_id* внутри селектора | В селекторах в поле для *hex\_id* появляется *id* гексагона | Рису-нок 4.1 |

Продолжение таблицы 4.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *1* | 2 | 3 | 4 | 5 | 6 | 7 |
| *D*-2 | При вводе некорректного *id* должно приходить уведомление | Сред­няя | 1. Ввести некорректный *ID* для гексагона | Нет уведомления о неправильном вводе *id* гексагона | В верхней части экрана должно появится уведомление о некорректности формата *hex*\_*id* | Рису-нок 4.2 |
| *D*-3 | Пользователь зафиксировал гексагон на карте, после чего сменил резолюцию | Средняя | 1. Выбрать любой гексагон на карте (через селекторы или через интерфейс) 2. Сменить резолюцию гексагона через селекторы | Точка отображения центра гексагона остается на координатах центра гексагона с меньшей резолюцией | 1. Отображение отметки на карте выбранного гексагона 2. Обновление карты после смены резолюции   Отсутствует метка зафиксированного гексагона | Рису-нок 4.3 |

На рисунке 4.1 демонстрируется ошибка отсутствия отображения *id* гексагона при выборе шестиугольника через интерфейс гексагональной карты.



Рисунок 4.1 – Отличия между *id* гексагонов

На рисунке 4.2 демонстрируется ошибка отсутствия уведомления после неправильного ввода *id* гексагона.

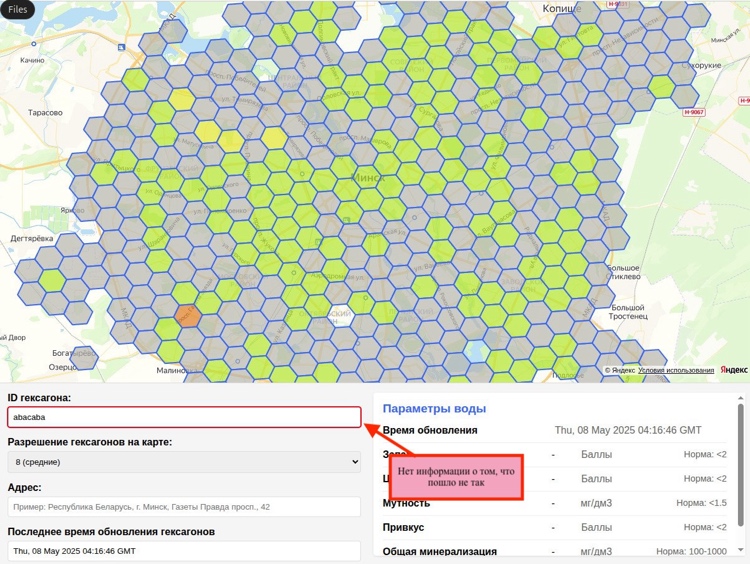


Рисунок 4.2 – Отсутствие уведомления после неправильного ввода *id* гексагона

На рисунке 4.3 демонстрируется ошибка расположения отметки центра гексагона после смены резолюции.



Рисунок 4.3 – Ошибка расположения отметки центра гексагона после смены резолюции

В рамках проведения тестирования веб-приложения выполнены комплексные проверки функциональности, пользовательского интерфейса и стабильности работы системы.

Тестирование проводилось в средах разработки *Visual Studio Code*  и *PyCharm,* использовались веб-браузеры *Google Chrome* и *Mozila Firefox*. Также при тестировании использовался сервер, развернутый на локальной машине, и локальная база данных *Postgres*.

В результате выполнения функционального тестирования был составлен отчет, представленный в таблице 4.3.

Таблица 4.3 – Отчет о выполнении функционального тестирования.

|  |  |
| --- | --- |
| Общая оценка качества | *Smoke* тестирование: пройдено |
| Качество приложения: высокое |
| Рекомендации по модулям | |
| Модуль | Рекомендации |
| Главная страница | Обратить внимание на правописание и стилистическое оформление селекторов. |
| Селекторы | Обратить внимание на заполняемся полей в селекторах. При выборе другого селектора предыдущей селектор должен переходить в изначальное состояние. При неправильном вводе селектора необходимо сообщать об этом пользователю. |
| Рекомендации | |
| В приложении выявлены незначительные недочеты, не блокирующие его работу. Среди них –отсутствие уведомления о некорректном вводе *id* гексагона и отсутствие отображения *id* гексагона в селекторах. Наиболее важной проблемой стала ошибка расположения отметки центра гексагона после смены резолюции, что затрудняет понимание пользователем текущего расположения гексагона на карте. Исправление этой ошибки является приоритетным. Устранение всех выявленных недостатков повысит качество интерфейса и обеспечит корректное функционирование приложения. | |

По итогам тестирования выявленные дефекты были исправлены. Результаты тестирования показали, что приложение готово к использованию в реальных условиях. Оно успешно решает задачи усовершенствования визуализации мониторинга качества воды. В дальнейшем возможно расширение функционала приложения, включая интеграцию с уведомлением пользователей и поддержку более сложных сценариев агрегирования показателей воды, что сделает его еще более универсальным и полезным для конечного пользователя [23].

Таким образом, функциональное тестирование подтвердило, что приложение соответствует всем предъявляемым требованиям и готово к внедрению в производственный процесс.

# 5 ТЕХНИКО-Экономическое обоснование разработки и использования Веб-приложениЯ по усовершенствованию визуализации мониторинга качества воды посредством введения цветовых гексагонов на карте города Минска

## 5.1 Характеристика разработанного веб-приложения по индивидуальному заказу

Данный раздел посвящен экономическому обоснованию дипломного проекта на тему «Веб-приложение для усовершенствованной визуализации мониторинга качества воды с использованием цветовых гексагонов на карте города Минска». Проект относится к категории разработок по индивидуальному заказу и направлен на повышение эффективности анализа и представления данных о качестве воды в городской среде. Так как разработка программного обеспечения ведется для сторонней организации, для экономического обоснования выбрана методика обоснования разработки по индивидуальному заказу.

Основная цель разработки — улучшение визуализации данных мониторинга качества воды за счет интерактивной карты с цветовым кодированием в форме гексагонов. Это позволит выявлять проблемные зоны, упростит анализ экологической обстановки и повысит наглядность представления информации.

Текущий формат отображения качества воды состоит из карты, с отметкой местоположения в виде пина на карте, а также таблицы с различными характеристиками воды. Такой формат предполагает, что пользователь обладает достаточной информацией о предельно допустимых значениях, может сравнивать и оценивать риски тех или иных параметрах для здоровья. Разрабатываемое приложение позволит автоматизировать этот процесс для конечного пользователя в следствие использования цветовой разметки.

Область применения проекта — экология и охрана окружающей среды. Основными пользователями приложения будут граждане, интересующиеся текущей обстановкой качеств воды в Минске. Для пользователей приложение станет удобным инструментом для оценки качества воды в свой районе.

Основным конкурентным преимуществом приложения является цветовое отображение по областям города Минска с помощью гексагонов на карте. Помимо этого, приложение поддерживает историческую информацию о данных: можно отобразить на карте данные за конкретный период времени.

Экономическая оценка целесообразности инвестиций в разработку и использование программного средства осуществляется на основе расчета и оценки следующих показателей: чистый дисконтированный доход, рентабельность инвестиций и простой срок окупаемости инвестиций.

## 5.2 Расчет основных затрат на разработку

**5.2.1** Расчет затрат на основную заработную плату разработчикам.

Для расчета затрат на разработку программного средства в первую очередь необходимо рассчитать основную заработную плату команды разработчиков. Расчет осуществляется исходя из состава и численности команды, размера месячной заработной платы каждого участника команды, а также трудоемкости работ, выполняемых при разработке программного средства отдельными исполнителями по формуле:

(5.1)

|  |  |  |  |
| --- | --- | --- | --- |
| где | Кпр | − | коэффициент премий (равный 1,5); |
|  | *n* | − | категории исполнителей, занятых разработкой программного средства; |
|  | Зчi | − | часовая заработная плата исполнителя *i*-й категории, р.; |
|  | *ti* | − | трудоемкость работ, выполняемых исполнителем *i*-й категории, определяется исходя из сложности разработки программного обеспечения и объема выполняемых им функций, ч. |

На 2025 год расчетная норма рабочего времени для пятидневной рабочей недели составляет 168 часов, 8 часов работы в день, среднемесячная расчетная норма рабочего времени – 21 день.

В разработке были задействованы следующие сотрудники: бизнес-аналитик, программист, тестировщик. Часовая заработная плата каждого сотрудника определена как результат деления месячной заработной платы (оклада) на количество рабочих часов в месяце. Расчет основных затрат представлен в таблице 5.1.

Таблица 5.1 ‒ Расчет затрат на основную заработную плату разработчиков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория исполнителя | Месячный оклад, р | Часовой оклад, р | Трудоемкость работ, Р | Итого, р |
| Бизнес-аналитик | 1463 | 9 | 25 | 225 |
| Программист | 2091 | 12 | 450 | 5400 |
| Тестировщик | 1826 | 10 | 120 | 1200 |
| Итого | | | | 6825 |
| Премия и иные стимулирующие выплаты (50%) | | | | 3412 |
| Основная заработная плата разработчиков | | | | 10237 |

**5.2.2** Расчет затрат на дополнительную заработную плату разработчикам

Дополнительная заработная плата ‒ это оплата за сверхурочный труд, различные трудовые успехи и надбавки за особые условия труда команды и включает выплаты, предусмотренные законодательством о труде, и определяется по нормативу в процентах (составляет 20%) к основной заработной плате по следующей формуле:

(5.2)

где Зо ‒ затраты на основную заработную плату;

Нд ‒ норматив дополнительной заработной платы, 20%.

Подставим значение в формулу (5.2) и вычислим Зд:

Согласно расчетам, затраты на дополнительную заработную плату разработчикам составит 2047 рубля.

**5.2.3** Расчет отчислений на социальные нужды

В расчете отчислений на социальные нужды учитываются обязательные платежи по установленным законодательством тарифам в фонд социальной защиты населения, а также затраты предприятия на обязательное социальное медицинское страхование для определенных категорий работников в соответствии с законодательством. Размер отчислений в фонд социальной защиты населения и на обязательное страхование определяется в соответствии с действующими законодательными актами Республики Беларусь и рассчитывается по установленной формуле:

(5.3)

где Нсоц ‒ норматив отчислений на социальные нужды, %.

Согласно законодательству Республики Беларусь норматив отчислений на социальные нужды составляет 34,6 %. Подставим результаты вычислений в формулу (5.3) и вычислим Рсоц:

Согласно расчетам, размер отчислений в фонд социальной защиты и на обязательное страхование составляет рублей.

**5.2.4** Расчет затрат на прочие расходы

Прочие расходы связаны с функционированием организации-разработчика в целом, например: затраты на аренду офисных помещений, отопление, освещение, амортизацию основных производственных фондов и так далее. При расчете данной статьи затрат учитывается норматив прочих затрат в целом по организации. В данном случае норматив прочих затрат равен 20 %. Размер затрат на прочие расходы рассчитывается по формуле:

(5.4)

где Ннз ‒ норматив прочих затрат в целом по организации, 30 %.

Подставим значение из выражения в формулу (5.4) и произведем расчет Рпр:

Согласно расчетам, размер затрат на прочие расходы составляет 3071 рубля.

**5.2.5** Расчет суммы затрат на разработку

Общая сумма затрат на разработку рассчитывается путем суммирования основной заработной платы, дополнительной заработной платы, отчислений на социальные нужды, прочих затрат. Формула расчета имеет следующий вид:

(5.5)

Подставим результаты вычислений в формулу (5.5) и произведем расчет Зр:

Согласно расчетам, сумма затрат на разработку составляет рубля.

**5.2.6** Расчет плановой прибыли, включаемой в цену программного средства

Плановая прибыль, включаемая в цену программного средства, рассчитывается по формуле:

(5.6)

В данном случае рентабельность затрат на разработку программного средства установили на уровне 25%. Подставим значение из выражения в формулу (5.6) и произведем расчет Пп.с.:

Исходя из расчетов, плановая прибыль, включаемая в цену программного средства, составляет рублей.

**5.2.7** Расчет отпускной цены программного средства

Отпускная цена программного продукта представляет собой сумму затрат на заработную плату и плановой прибыли. Рассмотрим формулу расчета отпускной цены программного средства:

(5.7)

Подставим результат вычислений и произведем расчет Цп.с.:

Исходя из расчетов, отпускная цена программного средства составляет рублей.

**5.2.8** Результаты расчета затрат на разработку и цены приложения для анализа рабочего времени и показателей производительности водителей крупногабаритных транспортных средств

В данном подразделе были рассчитаны необходимые статьи для расчета затрат на разработку и для расчета цены программного средства, а именно: основная заработная плата разработчиков, дополнительная заработная плата разработчиков, отчисления на социальные нужды, прочие расходы и плановая прибыль. Результаты расчетов представлены в таблице 5.2.

Таблица 5.2 ‒ Результаты расчета цены на разработку программного средства

|  |  |
| --- | --- |
| Наименование статьи затрат | Сумма, р. |
| Основная заработная плата разработчиков | 10237 |
| Дополнительная заработная плата разработчиков | 2047 |
| Отчисления на социальные нужды | 4176,5 |
| Прочие расходы | 3071 |
| Всего затраты на разработку | 19531,5 |
| Плановая прибыль | 4882,8 |
| Цена программного средства | 24414,3 |

## 5.3 Расчет результата от разработки и реализации веб-приложения

Для организации-разработчика экономическим эффектом является прирост чистой прибыли, полученной от разработки и реализации программного средства заказчику. Так как программное средство будет реализовываться организацией-разработчиком по отпускной цене, сформированной на основе затрат на разработку, то экономический эффект, полученный организацией-разработчиком, в виде прироста чистой прибыли от его разработки, определяется по формуле:

(5.8)

где Пп.с. ‒ прибыль, включаемая в цену программного средства, р;

Нп   ‒ ставка налога на прибыль согласно действующему законодательству, (по состоянию на 01.01.2024 г. – 20%).

Подставим результат вычисления в формулу (5.8) и произведем расчет ΔПч:

Исходя из расчетов, экономический эффект составляет 3906 рублей.

Для организации-заказчика расчет экономического эффекта от использования программного обеспечения, разработанного по индивидуальному заказу сторонней организацией, осуществляется в соответствии с методикой расчета основных видов экономического эффекта.

Экономия на заработной плате и начислениях на заработную плату сотрудников за счет снижения трудоемкости работ определяется по формуле:

(5.9)

|  |  |  |  |
| --- | --- | --- | --- |
| где | Кпр | – | коэффициент премий (по фактическим данным предприятия или в диапазоне 1,5-2); |
|  | *tр*без п.с | – | трудоемкость выполнения работ сотрудниками до внедрения веб-приложения, ч; |
|  | *t*рс п.с | – | трудоемкость выполнения работ сотрудниками после внедрения веб-приложения, ч; |
|  | *Т*ч | – | часовой оклад (часовая тарифная ставка) сотрудника, использующего программное средство, р; |
|  | *N*п | – | плановый объем работ, выполняемых сотрудником |
|  | Нд | – | норматив дополнительной заработной платы; |
|  | Нсоц | – | ставка отчислений от заработной платы, включаемых в себестоимость (34,6%). |

Подставим результат вычисления в формулу (5.9) и произведем расчет Эз.п:

Экономия на заработной плате и начислениях на заработную плату в результате сокращения численности работников определяется по формуле:

(5.10)

|  |  |  |  |
| --- | --- | --- | --- |
| где | *n* | – | категории работников, высвобождаемых в результате внедрения программного средства; |
|  | ΔЧ*i* | – | численность работников *i*-й категории, высвобожденных после внедрения программного средства, чел.; |
|  | З*i* | – | годовая заработная плата высвобожденных работников *i*-й категории после внедрения программного средства, р.; |
|  | Нсоц | – | норматив отчислений от заработной платы в соответствии с законодательством, %. |

Экономия на заработной плате и начислениях на заработную плату в результате сокращения численности работников составляет 0 р., поскольку количество сотрудников не изменилось после внедрения веб-приложения.

Экономия на материальных ресурсах в результате внедрения программного средства определяется по формуле:

(5.11)

|  |  |  |  |
| --- | --- | --- | --- |
| где | Кт.р | – | коэффициент транспортных расходов (по данным предприятия или 1,05-1,2); |
|  | Нрп.с | – | норма расхода материальных ресурсов при выполнении работ сотрудниками до и после внедрения программного средства, нат. ед. [27]; |
|  | Цм | – | цена за единицу материального ресурса, р.; |
|  | *N*п | – | плановый объем работ, выполняемых сотрудником с использованием программного средства. |

Экономия на материальных ресурсах так же равно 0 р., поскольку расход материальных ресурсов не изменился.

Экономическим эффектом при использовании программного средства является прирост чистой прибыли, полученной за счет экономии на текущих затратах предприятия, который рассчитывается по формуле:

(5.12)

|  |  |  |  |
| --- | --- | --- | --- |
| где | Этек | – | экономия на текущих затратах при использовании программного средства, р.; |
|  | ΔЗтекп.с | – | прирост текущих затрат, связанных с использованием программного средства, р; |
|  | Нп | – | ставка налога на прибыль согласно действующему законодательству. |

Таким образом экономический эффект при использовании программного средства составит

## 5.4 Расчет показателей экономической эффективности разработки и использования веб-приложения

Для организации-разработчика программного средства оценка экономической эффективности разработки осуществляется с помощью расчета рентабельности затрат на разработку программного средства. Рентабельность является одним из основных показателей эффективности предприятия с точки зрения использования привлеченных средств. Она представляет собой отношение суммы чистой приведенной прибыли, полученной за весь расчетный период, к суммарным приведенным затратам за этот же период и определяется по формуле:

(5.13)

|  |  |  |  |
| --- | --- | --- | --- |
| где | ΔПч | – | прирост чистой прибыли, полученной от разработки программного средства организацией-разработчиком по индивидуальному заказу, р; |
|  | Зр | – | затраты на разработку программного средства организацией-разработчиком, р. |

Подставим результат вычисления в формулу (5.13) и произведем расчет Ри:

Рассчитанный показатель отображает, сколько чистой прибыли компания-разработчик получит от вложенных денег в разработку программного средства.

Так как сумма инвестиций больше суммы годового прироста чистой прибыли, для организации-заказчика рассчитывается несколько показателей экономической эффективности.

Для приведения доходов и затрат к настоящему моменту времени определяется коэффициент дисконтирования по формуле:

(5.14)

|  |  |  |  |
| --- | --- | --- | --- |
| где | *d* | – | требуемая норма дисконта, которая по своему смыслу соответствует устанавливаемому инвестором желаемому уровню рентабельности инвестиций, доли единицы; |
|  | *t* | – | порядковый номер года, доходы и затраты которого приводятся к расчетному году; |
|  | *tp* | – | расчетный год, к которому приводятся доходы и инвестиционные затраты. |

Норму дисконта принимаем равным ставке рефинансирования Национального банка Республики Беларусь – 9,5%. Расчетный период составит четыре года.

Таким образом, коэффициенты дисконтирования за каждый год составляют:

В течение первого года осуществляется разработка приложения, поэтому в первый год экономический эффект будет меньше планируемого. Так как работа команды разработчиков осуществляется поэтапно, то затраченное время будет равно сумме трудоемкости работ команды, и составит 595 часов. В таблице 5.3 приведен расчет эффективности инвестиций (затрат) в реализацию проектного решения.

Таблица 5.3 – Расчет эффективности инвестиций (затрат) в реализацию проектного решения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Показатель | Значение расчетного периода по годам | | | |
| 1-й год | 2-й год | 3-й год | 4-й год |
| Прирост чистой прибыли, р. | 12997,7 |  |  |  |
| Дисконтированный резуль­тат, р | 12997,7 | 14602,4 | 13318,7 | 12195,4 |
| Инвестиции в разработку, р. | 19531,5 | 0 | 0 | 0 |
| Дисконтированные инвестиции, р. | 19531,5 | 0 | 0 | 0 |
| Чистый дисконтированный доход по годам, р. | -6656,3 | 14602,4 | 13318,7 | 12195,4 |
| Чистый дисконтированный доход нарастающим итого, р. | -6656,3 | 7946,1 | 21264,8 | 33460,2 |
| Коэффициент дисконтирования, доли единицы | 1,00 | 0,91 | 0,83 | 0,76 |

В данном случае дисконтированный эффект нарастающим итогом превысит дисконтированные инвестиции на второй год. Простой срок окупаемости рассчитывается по формуле:

(5.15)

Таким образом, простой срок окупаемости инвестиций равен:

Чистый дисконтированный доход рассчитывается по формуле:

(5.16)

Таким образом, чистый дисконтированный доход равен:

В результате проведения расчетов была определена необходимость разработки программного обеспечения, а также получен экономический эффект от использования данного программного продукта. По результатам проведенного экономического обоснования были получены следующие результаты:

1 Стоимость заказа на разработку составила 24414,3 рублей.

2 Прирост чистой прибыли составил 16046,6 рублей.

3 Данная разработка имеет положительный экономический эффект в размере 82%.

4 По результатам проведенных расчетов, вложенные инвестиции должны окупится за 1,28 года, чистую прибыль стоит ожидать на второй год реализации проекта.

Таким образом, разработка и реализация по индивидуальному заказу веб-приложения для усовершенствованной визуализации мониторинга качества воды с использованием цветовых гексагонов на карте города Минска с экономической точки зрения целесообразна.

# 6 ОХРАНА ТРУДА. МЕРОПРИЯТИЯ ПО ОБЕСПЕЧЕНИЮ ВЫСОКОЙ РАБОТОСПОСОБНОСТИ И СОЗДАНИЮ КОМФОРТНЫХ УСЛОВИЙ ТРУДА ПРИ РАЗРАБОТКЕ И ЭКСПЛУАТАЦИИ ВЕБ-ПРИЛОЖЕНИЯ

Работа операторов, занимающихся разработкой веб-приложений, требует непрерывного взаимодействия с различными устройствами, такими как мониторы, ноутбуки, планшеты и другие средства отображения информации. Однако, при такой работе оператор может столкнуться с проблемами зрительного восприятия информации и формирования утомления зрительного анализатора, что в свою очередь может привести к ухудшению эффективности труда и появлению заболеваний глаз. Зрительное восприятие информации является сложным процессом, включающим в себя множество факторов, таких как размер и форма объекта, контрастность, яркость и цветность изображения, частота мерцания экрана. В процессе работы с СОИ оператор непрерывно воспринимает информацию, что может привести к утомлению зрительного анализатора и снижению эффективности работы [28]. Утомление зрительного анализатора проявляется в виде ухудшения качества восприятия информации, затруднения в концентрации внимания, увеличения количества ошибок при выполнении задач. Чтобы избежать таких последствий, необходимо учитывать особенности зрительного восприятия при разработке и уделять внимание организации рабочего пространства [29]. Среди главных причин утомления зрительного анализатора выделяют: низкое качество изображения на мониторе, плохая контрастность, размытость, искажения цветопередачи, неравномерная яркость. Все это приводит к тому, что глаза оператора вынуждены работать с большей интенсивностью, чтобы воспринять информацию.

Недостаточная или избыточная яркость освещения рабочего пространства. Отсутствие достаточного освещения или наличие слишком яркого света может привести к резкому контрасту между светом и монитором, что ухудшит качество восприятия информации и приведет к быстрому утомлению глаз.

Отсутствие возможности регулирования яркости и контрастности монитора. В процессе разработки веб-приложения некоторые изображения, отображаемые на экране, могут быть слишком яркими или слишком темными, что также может ухудшить качество восприятия информации и привести к утомлению глаз.

Неправильное расположение экрана и рабочего места. Если экран находится на неправильной высоте или на неправильном расстоянии от глаз оператора, это может привести к увеличению нагрузки на мышцы глаз и шеи, что в свою очередь может привести к болезненным ощущениям и утомлению.

Монотонность работы. Длительное время, проведенное за монитором, без перерывов и смены задач, может привести к утомлению зрительного анализатора и снижению эффективности работы оператора [30].

Одним из способов решения проблем, вызывающих проблемы со здоровьем у операторов, является соблюдение инженерно-психологических требований − специальных правил и рекомендаций, разработанных с учетом особенностей человеческого организма. Они помогают создать наиболее оптимальные условия для работы с информацией, минимизируя возможные проблемы со здоровьем. Такие требования могут предъявляться к различным системам и устройствам, используемым оператором, таким как средства отображения информации, устройства управления, средства связи [31]. Чтобы избежать проблем со зрительным восприятием и утомления глаз, необходимо учитывать следующие инженерно-психологические требования к средствам отображения информации и их расположению в рабочем пространстве.

Разрешение экрана. Высокое разрешение экрана позволяет получать четкие и детализированные изображения, что снижает утомляемость глаз. Оптимальным разрешением экрана считается 1920x1080 пикселей.

Мерцание экрана. Операторам рекомендуется выбирать мониторы с минимальным уровнем мерцания экрана. Оптимальный вариант - мониторы с частотой обновления экрана не менее 60 Гц.

Подсветка экрана. Экран должен быть равномерно подсвечен, без ярких и темных пятен. Также следует учитывать уровень яркости экрана и его контрастность. Зачастую производители мониторов предоставляют специальные шаблоны настройки яркости, контрастности и цвета для работы в разных условиях, таких как дневное светлое время, темное время суток или различные условия освещения, однако такие настройки не всегда могут соответствовать требованиям конкретного оператора. Таким образом, 82 оптимальный уровень яркости экрана зависит от типа работы и задач, выполняемых пользователем, а также от условий окружающей среды и настраивается индивидуально для каждого оператора.

Размер шрифта. Размер шрифта должен быть достаточно большим для легкого чтения, но не слишком крупным, чтобы не загружать глаза. Рекомендуемый размер шрифта − от 10 до 14 пунктов. Однако следует учитывать, что оптимальный размер шрифта может изменяться от конкретной отображаемой информации и особенностей зрительного восприятия оператора.

Расположение экрана. Экран должен быть расположен на уровне глаз, чтобы оператор мог смотреть на него без необходимости наклонять или поворачивать голову. Наклон экрана должен быть примерно 15 градусов, чтобы уменьшить напряжение в шее и спине. Расстояние от глаз до экрана также имеет значение. Если экран расположен слишком близко, то глаза будут напряжены, что может привести к усталости глаз и боли в голове. Если же экран расположен слишком далеко, то глаза будут напряжены, чтобы читать текст на экране, что также может привести к усталости глаз и боли в голове. Оптимальное расстояние от экрана до глаз составляет от 50 до 70 см.

Качество монитора. Мониторы с качественной матрицей, такой как *IPS* (*In-Plane Switching)* или *PLS* (*Plane-to-Line Switching*), обеспечивают более четкое и точное отображение изображений и цветов. Они также имеют более широкий угол обзора, что позволяет смотреть на экран с разных углов без потери качества изображения. *IPS* и *PLS* матрицы также обеспечивают более равномерную яркость на всей поверхности экрана и не имеют мерцания, что помогает снизить утомляемость глаз [32].

При разработке веб-приложения важно также учесть безопасность и удобство взаимодействия пользователей с приложением. Веб-приложение может использоваться в различных условиях и на различных устройствах. Поэтому разработчикам важно предусмотреть возможность настройки интерфейса приложения, чтобы пользователи могли изменить размеры и шрифты в соответствии с их индивидуальными потребностями. Важно также учитывать возможность адаптации веб-приложения для пользователей с ограниченными возможностями зрения, таких как люди с дальнозоркостью, близорукостью, цветовой слепотой. Например, для людей с цветовой слепотой необходимо предусмотреть возможность отображения информации в режиме повышенной контрастности.

Ключевую роль в зрительном восприятии информации играет освещение рабочего пространства. Освещение должно соответствовать определенным требованиям.

Интенсивность освещения. Оптимальный уровень освещенности рабочей поверхности должен быть не менее 500 лк, однако, в зависимости от характера работы, этот уровень может быть увеличен до 1000 лк или более. Равномерность освещения. Освещение должно быть равномерно по всей рабочей поверхности. При наличии ярких и темных пятен на экране или рабочей поверхности, глаза оператора быстро устают.

Тип и цвет света. Рекомендуется использовать лампы с природным белым светом (4000-4500 К) или дневным светом (5000-5500 К). Не рекомендуется применение ламп с желтым светом, которые могут вызывать усталость глаз.

Блики и отражения. Необходимо избегать бликов на экране и отражений от других объектов, таких как стены, окна или предметы на рабочей поверхности. Это можно достичь за счет расположения рабочего места под правильным углом и использования плотных штор или жалюзи для блокировки прямых лучей солнца.

Дополнительное освещение. При необходимости можно использовать дополнительное освещение, такое как настольные лампы или светодиодные ленты, чтобы улучшить видимость на определенных участках рабочей поверхности [33].

Еще одним важным фактором, влияющим на здоровье и утомляемость зрительного анализатора, является режим труда и отдыха оператора. Основные рекомендации в этой области включают разнообразие задач. Операторам следует предоставлять разнообразные задачи для выполнения, чтобы избежать монотонной работы и уменьшить утомляемость. Регулярные перерывы помогают снизить усталость и напряжение глаз, а также уменьшить риск возникновения мышечной травмы. Рекомендуется делать перерывы каждый час на 10-15 минут.

Физические упражнения, упражнения для глаз помогают уменьшить усталость и напряжение в мышцах.

Регулярные проверки зрения. Операторы должны регулярно проверять зрение, чтобы выявить возможные проблемы и предотвратить их развитие.

Правильная организация рабочего места. Рабочее место должно быть организовано таким образом, чтобы минимизировать напряжение и усталость оператора. Это может включать в себя правильное расположение средств отображения информации, использование эргономичных кресел и столов, а также правильное расположение клавиатуры и мыши.

Регулярное обслуживание оборудования помогает предотвратить неожиданные поломки и снизить риск травмы, связанной с работой с оборудованием, которое неисправно или не настроено правильно [34].

В рамках разработки дипломного проекта учтены все вышеперечисленные рекомендации. Был использован монитор с частотой 60 Гц и разрешением экрана 1920х1080 пикселей, расположенный на уровне глаз на расстоянии около 50 см. Яркость монитора настроена на оптимальный уровень, обеспечивающий четкое восприятие изображения. Размер шрифта варьировался в зависимости от текущих задач, однако всегда был достаточно крупным для комфортного восприятия информации без излишнего напряжения глаз. Разработка велась в дневное время при естественном дневном освещении. Для минимизации излишней яркости и отражений использованы жалюзи. В ходе разработки регулярно осуществлялись перерывы и смена задач для предотвращения монотонности труда. Таким образом, оптимизация зрительного взаимодействия оператора со средствами отображения информации при разработке веб-приложения является важным аспектом охраны труда. Требования к организации, качественным и количественным характеристикам освещения рабочего места оператора, а также оптимизация режима труда и отдыха оператора − все эти факторы влияют на здоровье и безопасность работы оператора при разработке веб-приложения.

# ЗАКЛЮЧЕНИЕ

В рамках дипломного проекта спроектировано и разработано веб-приложение для по усовершенствованию визуализации мониторинга качества воды.

Для достижения поставленной цели был проведён анализ предметной области, изучены существующие аналоги информационных систем, направленных на анализ, визуализацию и сбор информации о качестве воды.

В отличие от существующих решений по визуализации качества воды в Минске, разработанное веб-приложение предлагает интерфейс, содержащий информацию о качестве воды в виде покрытия тепловой гексагональной картой города Минска. Реализована возможность выбирать для визуализации разные промежутки времени.

Реализованы все функциональные требования, предъявленные к разрабатываемому веб-приложению Разработана архитектура программного средства, составлены алгоритмы работы пользователя, а также спроектирована и реализована база данных с использованием *PostgreSQL*.

В дипломном проекте используется следующий стек технологий:

*Backend* - серверная часть программного средства реализована с использованием языка программирования *Python*.

*Frontend* - клиентская часть программного средства реализована с помощью технологий *HTML*, *CSS*, *JavaScript*.

Для хранения данных программного средства использована реляционная база данных *PostgreSQL*.

Разработанное веб-приложение предоставляет пользователю следующие функции:

‒ просмотр качества воды по адресу через взаимодействие с картой;

‒ просмотр качества воды по адресу через ручной ввод;

‒ просмотр качества воды по гексагону через взаимодействие с картой;

‒ просмотр качества воды по гексагону через ручной ввод *hex\_id*;

‒ выбор даты, за которую необходимо сделать визуализацию.

К пояснительной записке прилагаются следующие графические материалы: структурная схема, структура базы данных, блок-схема алгоритма работы веб-приложения, диаграмма вариантов использования веб-приложения, эскизы рабочих окон, результаты расчета надежности веб-приложения.

В ходе выполнения расчета надежности веб-приложения были выполнены следующие пункты:

1 Проведена оценка надежности по модели сложности. В ходе выполнения расчета были вычислены размер и потенциальный объем веб-приложения, абсолютная и относительные сложности программы, максимальный уровень вложенности операторов условия, метрики Чепина. Для объектно-ориентированных характеристик рассчитаны метрики Чидамбера и Кемерера. По результатам оценки надежность программного средства составила 0,907.

2 Выполнен расчет надежности по модели Джелинского-Моранды, в ходе которого были определены вероятность безотказной работы веб-приложения, среднее время до появления следующей ошибки и время до окончания тестирования. По итогам расчетов вероятность безотказной работы составила 0,95.

3 Надежность веб-приложения, посчитанная по модели Муса, составила 0,87. Определена надежность веб-приложения для периода эксплуатации T, равному 128 часам, а также средняя наработка до отказа после тестирования на этапе эксплуатации веб-приложения.

Программное средство прошло тестирование: реализованы тест-кейсы, проведено функциональное и smoke-тестирование. Выявлены и устранены две ошибки уровня среднего уровня и одна ошибка высокого уровня, после чего повторное тестирование показало полное соответствие заявленным требованиям.

Выполнено технико-экономическое обоснование разработки. Получены следующие результаты:

‒ прирост чистой прибыли для заказчика при использовании системы составит 12997,7 рубля;

‒ срок окупаемости проекта — 1 год и 4 месяца;

‒ чистый дисконтированный доход за 4 года эксплуатации — рубля.

В разделе «Охрана труда. Мероприятия по обеспечению высокой работоспособности и созданию комфортных условий труда при разработке и эксплуатации веб-приложения» предложены мероприятия по снижению утомляемости разработчиков: организация оптимального светового режима, перерывы при работе за ПК, использование эргономичных рабочих мест.

Результаты разработанного программного обеспечения представлены в сборнике материалов 61 научно-практической конференции. Данный дипломный проект выполнен мной лично, проверен на заимствования, процент оригинальности составляет 75,37% (отчет о проверке на заимствования прилагается).

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Минскводоканал [Электронный ресурс]. – Режим доступа: https://www.minskvodokanal.by. – Дата доступа: 12.02.2025.

[2] Uber H3 [Электронный ресурс]. – Режим доступа: https://github.com/uber/h3. – Дата доступа: 20.02.2025.

[3] Почему стоит выбрать PostgreSQL [Электронный ресурс]. – Режим доступа: https://cloud.ru/blog/postgresql-chto-eto-za-subd. – Дата доступа: 01.03.2025.

[4] Качество воды в вашем доме. Технологический и лабораторный контроль [Электронный ресурс]. – Режим доступа: https://minskvodokanal.by/water/home. – Дата доступа: 02.03.2025.

[5] Использование Cron задач для регулярных процессов [Электронный ресурс]. – Режим доступа: https://www.linode.com/docs/guides/schedule-tasks-with-cron. – Дата доступа: 15.02.2025.

[6] Water Quality Map [Электронный ресурс]. – Режим доступа: https://waterqualitymap.org. – Дата доступа: 15.02.2025.

[7] Wasserkiez [Электронный ресурс]. – Режим доступа: https://www.wasserkiez.de. – Дата доступа: 18.02.2025.

[8] Uber H3 Viewer [Электронный ресурс]. – Режим доступа: https://wolf-h3-viewer.glitch.me. – Дата доступа: 20.02.2025.

[9] JavaScript API Яндекс Карт. Версия 3.0 [Электронный ресурс]. – Режим доступа: https://yandex.ru/maps-api/docs/js-api/index.html. – Дата доступа: 22.02.2025.

[10] h3-js [Электронный ресурс]. – Режим доступа: https://github.com/uber/h3-js. – Дата доступа: 24.02.2025.

[11] Интеграция Python и JavaScript [Электронный ресурс]. – Режим доступа: https://medium.com/swlh/run-python-script-from-js-and-send-data-to-browser-15677fcf199f. – Дата доступа: 25.02.2025.

[12] Flask [Электронный ресурс]. – Режим доступа: https://flask.palletsprojects.com/en/stable. – Дата доступа: 27.02.2025.

[13] Selenium [Электронный ресурс]. – Режим доступа: https://www.selenium.dev. – Дата доступа: 03.03.2025.

[14] Schedule [Электронный ресурс]. – Режим доступа: https://schedule.readthedocs.io/en/stable/index.html. – Дата доступа: 05.03.2025.

[15] Cross-Origin Resource Sharing [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/HTTP/Guides/CORS. – Дата доступа: 08.03.2025.

[16] Статья про написания Cron задач [Электронный ресурс]. – Режим доступа: https://ljinlabs.medium.com/scheduling-python-scripts-with-crontab-8cd6a03c5598. – Дата доступа: 28.02.2025.

[17] API Я.Геокодера [Электронный ресурс]. – Режим доступа: https://yandex.ru/maps-api/docs/geocoder-api/index.html. – Дата доступа: 05.03.2025.

[18] HSV [Электронный ресурс]. – Режим доступа: https://medium.com/@venkatpendela96/rgb-vs-hsv-52e55e504120. – Дата доступа: 08.03.2025.

[19] RGB [Электронный ресурс]. – Режим доступа: https://practicum.yandex.ru/blog/chto-takoe-cvetovoe-prostranstvo-podrobnyj-razbor/. – Дата доступа: 08.03.2025.

[20] Распределение и масштабирование слоя хранения данных [Электронный ресурс]. – Режим доступа: <https://systems.education/architecting-distributed-transactional-apps-storage>. – Дата доступа:10.03.2025

[21] Собственные типы в PostgreSQL [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/docs/current/sql-createtype.html– Дата доступа: 10.03.2025.

[21] Осипович, В. С. Проектирование информационных систем промышленной безопасности. Курсовое проектирование : пособие / В. С. Осипович, В. В. Егоров. – Минск: БГУИР, 2020. – 84 с.: ил.

[22] Метрики сложности ПО [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/308594/. – Дата доступа: 11.03.2025.

[23] Общая математическая модель проявления ошибок в ПО. Гипотеза Джелинского-Моранды [Электронный ресурс]. – Режим доступа: https://clck.ru/3M7jNy. – Дата доступа: 07.03.2025.

[24] Модели оценки надежности программного обеспечения [Электронный ресурс]. – Режим доступа: https://cyberleninka.ru/article/n/modeli-otsenki-nadezhnosti-programmnogo-obespecheniya. – Дата доступа: 09.03.2025.

[25] Тест кейсы [Электронный ресурс]. – Режим

доступа: https://sky.pro/media/kak-napisat-test-keys. – Дата доступа: 09.03.2025.

[26] Функциональное тестирование ПО [Электронный ресурс]. – Режим доступа: https://practicum.yandex.ru/blog/funkcionalnoe-testirovanie-po Дата доступа: 09.03.2025.

[27] Республика Беларусь. Налоговый кодекс [Электронный ресурс]. – Режим доступа: https://kodeksy.by/nalogovyy-kodeks-respubliki-belarus. – Дата доступа: 24.03.2025.

[28] Эргономика : учеб. пособие / О. Н.Чиченева. – Мосвка : Изд. Дом НИТУ «МИСиС», 2019. – 118 с.

[29] Эргономическое проектирование рабочего места оператора: практикум для студентов специальности 1-36 01 03 «Технологическое оборудование машиностроительного производства» / А. И. Кочергин, Т. Н. Бабак. – Минск: БНТУ, 2019. – 27 с.

[30] Гигиена компьютерного труда : учеб. пособие / Н. Г Коршунова – Благовещенск : Амурская ГМА, 2021г. – 93с.

[31] Психология труда, инженерная психология и эргономика 2-е изд., пер. и доп. Учебник для вузов / Г. Н. Солнцевой, А. А. Обознова. – Москва : Издательство Юрайт, 2023. – 661 с.

[32] Типы экранов [Электронный ресурс]. Режим доступа: https://4pda.to/2017/10/15/347387/.

[33] Охрана труда. Курс лекций: учеб.-метод. пособие / В. Н. Босок, А. Е. Кондраль. – Минск : БГСХА, 2021. – 155 с.

[34] Важность управления техническим обслуживанием [Электронный ресурс]. Режим доступа: https://cleverics.ru/digital/2023/10/vazhnostupravleniya-texnicheskim-obsluzhivaniem-zach

# ПРИЛОЖЕНИЕ А (обязательное) Отчет о проверке на заимствование

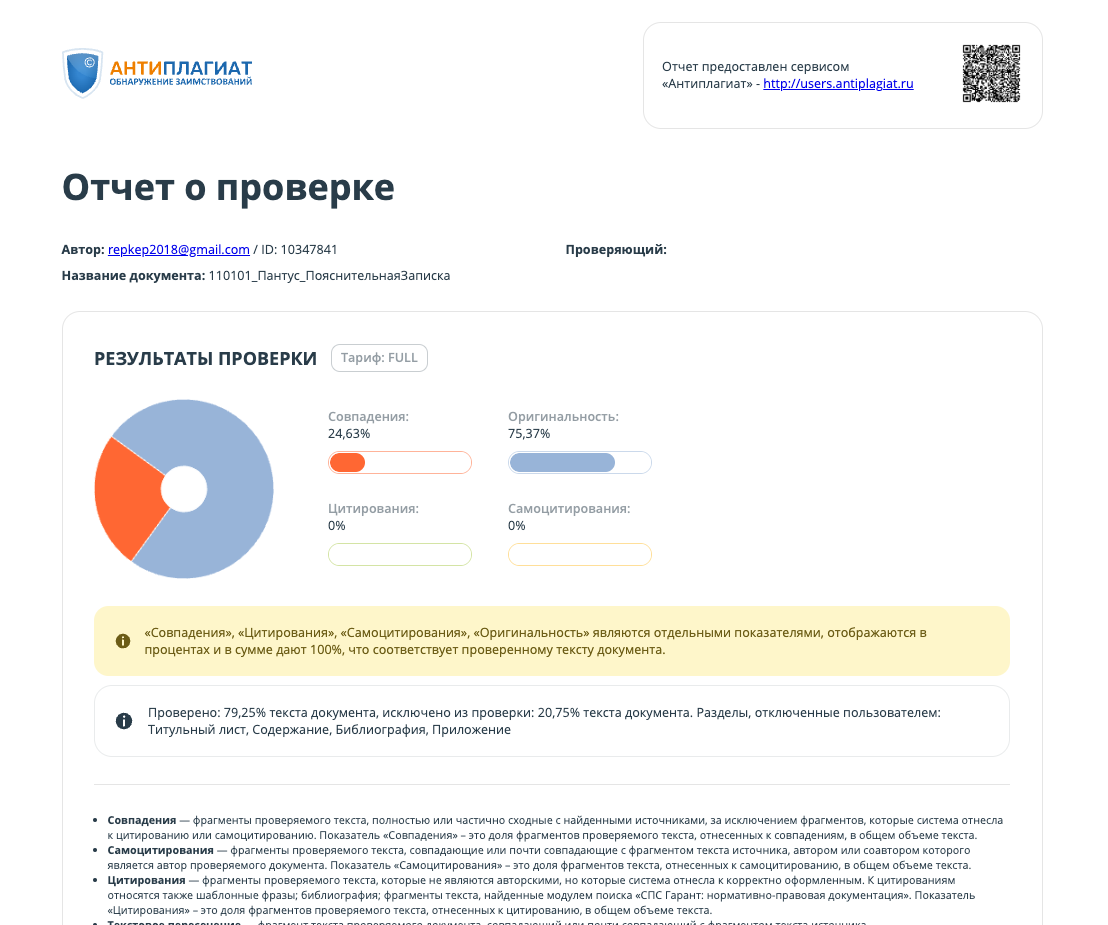
****

Рисунок А.1 – Результат проверки на заимствования в системе «Антиплагиат»

# Приложение Б (обязательное) Листинг программы

**Веб-приложение по усовершенствованию визуализации мониторинга качества воды посредством введения цветовых гексагонов на карте города Минска**

**Листинг программы**

**ГУИР.110101**

**Листов 24**

СОДЕРЖАНИЕ

[Б.1 КОНФИГУРАЦИЯ 81](#_Toc198047787)

[Б.2 ГЕОКОДИРОВАНИЕ 83](#_Toc198047788)

[Б.3 ВЗАИМОДЕЙСТВИЕ С БАЗОЙ ДАННЫХ 84](#_Toc198047789)

[Б.4 ПОЛУЧЕНИЕ АДРЕСОВ ЖИЛЫХ ДОМОВ МИНСКА 90](#_Toc198047790)

[Б.5 ПОЛУЧЕНИЕ ПОКАЗАТЕЛЕЙ ВОДЫ 92](#_Toc198047791)

[Б.5 ИСПОЛЬЗУЕМЫЕ МОДЕЛИ ДАННЫХ 95](#_Toc198047792)

[Б.6 РЕГИСТРАЦИЯ ОБРАБОТЧИКОВ 97](#_Toc198047793)

[Б.7 ОБНОВЛЕНИЕ ГЕО ИНФОРМАЦИИ 101](#_Toc198047794)

# Б.1 КОНФИГУРАЦИЯ

# Example of geo\_config.yaml

# map\_settings:

# allowed\_hexagons\_resolutions:

# - 7

# - 8

import os, yaml

from dataclasses import dataclass

@dataclass

class GeoConfig:

allowed\_hexagons\_resolutions: list[int]

def read\_geo\_config() -> GeoConfig:

dir\_path = os.path.dirname(os.path.realpath(\_\_file\_\_))

with open(f"{dir\_path}/geo.yaml", "r") as stream:

data\_loaded = yaml.safe\_load(stream)["map\_settings"]

return GeoConfig(

allowed\_hexagons\_resolutions=data\_loaded["allowed\_hexagons\_resolutions"],

)

def test\_read\_geo\_config():

cfg = read\_geo\_config()

assert cfg.allowed\_hexagons\_resolutions == [7, 8]

# Example of pg.yaml

# pg:

# dbname: some\_db

# username: some\_admin

# password: some\_password

import os, yaml

from dataclasses import dataclass

@dataclass

class PgConfig:

dbname: str

username: str

password: str

def read\_pg\_config() -> PgConfig:

def \_get\_port(port):

if port is not None:

return int(port)

return None

dir\_path = os.path.dirname(os.path.realpath(\_\_file\_\_))

with open(f"{dir\_path}/pg.yaml", "r") as stream:

data\_loaded = yaml.safe\_load(stream)["pg"]

return PgConfig(

dbname=data\_loaded["dbname"],

username=data\_loaded["username"],

password=data\_loaded["password"],

)

# Example of yandex.yaml

# yandex:

# geocoder:

# api\_key: "<YOUR\_API\_KEY>"

# requests\_limit: 0

# chunk\_request\_size: 10

import os, yaml

from dataclasses import dataclass

@dataclass

class GeocoderConfig:

api\_key: str

requests\_limit: int

chunk\_request\_size: int

def read\_geocoder\_config():

dir\_path = os.path.dirname(os.path.realpath(\_\_file\_\_))

with open(f"{dir\_path}/yandex.yaml", "r") as stream:

data\_loaded = yaml.safe\_load(stream)["yandex"]["geocoder"]

return GeocoderConfig(

api\_key=data\_loaded["api\_key"],

requests\_limit=int(data\_loaded["requests\_limit"]),

chunk\_request\_size=int(data\_loaded["chunk\_request\_size"]),

)

map\_settings:

allowed\_hexagons\_resolutions:

- 7

- 8

pg:

dbname: water\_heatmap\_db

username: water\_heatmap\_admin

password: repkep2018

yandex:

geocoder:

# api\_key: "52afd32b-2096-48e3-a8cf-8126cc58c620"

api\_key: "350d73aa-6e63-4ea4-a773-ac2baeadf011"

requests\_limit: 0

chunk\_request\_size: 10

# Б.2 ГЕОКОДИРОВАНИЕ

# pip install yandex-geocoder

# pip install h3

from decimal import Decimal

from yandex\_geocoder import Client

from external.config.yandex\_configs import GeocoderConfig, read\_geocoder\_config

from model.geo import Point

import h3

class GeocoderClient:

\_\_config: GeocoderConfig = read\_geocoder\_config()

\_\_geocoder\_client = Client(\_\_config.api\_key)

def coordinates(self, address: str):

lon, lat = self.\_\_geocoder\_client.coordinates(address)

return Point(

latitude=float(lat),

longitude=float(lon),

)

def address(self, point: Point):

return self.\_\_geocoder\_client.address(

Decimal(point.longitude), Decimal(point.latitude)

)

# convert with h3 library

def hexagon(self, address: str, hex\_resolution: int):

geo\_point = self.coordinates(address)

return h3.latlng\_to\_cell(

geo\_point.latitude, geo\_point.longitude, hex\_resolution

)

def hexagon\_to\_nearby\_house(self, h3\_cell: str):

lat, lon = h3.cell\_to\_latlng(h3\_cell)

return self.address(Point(latitude=lat, longitude=lon))

# Б.3 ВЗАИМОДЕЙСТВИЕ С БАЗОЙ ДАННЫХ

import datetime

import logging

import time

from typing import Optional

import h3

import psycopg2

from psycopg2.extras import register\_composite

from external.config.pg\_config import PgConfig, read\_pg\_config

from model.geo import Hexagon, AddressInfo, Point

from model.water\_parameters import WaterParameters, Parameter

def \_register\_custom\_types(conn):

register\_composite("parameter", conn)

register\_composite("water\_parameters", conn) # без этого не распарсим

register\_composite("color", conn)

register\_composite("geo\_point", conn)

class PgClient:

\_\_pg\_settings: PgConfig = read\_pg\_config()

def \_\_get\_db\_connection(self):

return psycopg2.connect(

dbname=self.\_\_pg\_settings.dbname,

user=self.\_\_pg\_settings.username,

password=self.\_\_pg\_settings.password,

)

def \_\_select\_query(self, query: str, \*kwargs):

conn = None

try:

logging.debug("Starting conn for SELECT")

conn = self.\_\_get\_db\_connection()

\_register\_custom\_types(conn)

with conn.cursor() as cursor:

cursor.execute(query, (\*kwargs,),)

result = cursor.fetchall()

return result

except Exception as e:

logging.error(f"Database error: {e}")

return None

finally:

if conn:

conn.close()

logging.debug("Connection for select closed")

def \_\_insert\_query(self, query: str, \*kwargs):

conn = None

try:

logging.debug("Starting conn for INSERT")

conn = self.\_\_get\_db\_connection()

\_register\_custom\_types(conn)

with conn.cursor() as cursor:

cursor.execute(query,(\*kwargs,),)

conn.commit()

except Exception as e:

logging.warning(f"Insertion error occurred {e}")

if conn:

conn.rollback()

raise

finally:

if conn:

conn.close()

logging.debug("Connection for insert closed")

def get\_address\_info(self, address: str) -> Optional[AddressInfo]:

query = "SELECT created\_at, address, coordinates, water\_parameters FROM address\_info WHERE address = %s ORDER BY created\_at DESC LIMIT 1"

result = self.\_\_select\_query(query, address)

assert len(result) < 2, 'will fail after multiple instances with diff created\_at\_ts'

return \_parse\_address\_info(result[0]) if result else None

"""

SELECT latest information about all addresses saved inside pg.

"""

def get\_all\_address\_info(self) -> list[AddressInfo]:

query = "SELECT DISTINCT ON (address) created\_at, address, coordinates, water\_parameters FROM address\_info ORDER BY address, created\_at DESC"

result = self.\_\_select\_query(query,)

return [\_parse\_address\_info(res) for res in result]

def get\_info\_about\_hex(self, hex\_id: str) -> Optional[Hexagon]:

query = "SELECT created\_at, hex\_id, hex\_resolution, hex\_color, avg\_water\_parameters FROM hexagons WHERE hex\_id = %s ORDER BY created\_at DESC LIMIT 1"

result = self.\_\_select\_query(query, hex\_id)

assert len(result) < 2, 'will fail after multiple instances with diff created\_at\_ts'

return \_parse\_hexagon(result[0]) if result else None

def get\_all\_hexes\_with\_res(self, hex\_resolution: int) -> list[Hexagon]:

query = "SELECT DISTINCT ON (hex\_id) created\_at, hex\_id, hex\_resolution, hex\_color, avg\_water\_parameters FROM hexagons WHERE hex\_resolution = %s ORDER BY hex\_id, created\_at DESC"

result = self.\_\_select\_query(query, hex\_resolution)

return [\_parse\_hexagon(res) for res in result]

def insert\_hexagon(

self,

hex\_id: str,

hex\_color: tuple[int, int, int],

water\_parameters: Optional[WaterParameters],

):

r, g, b = hex\_color

if water\_parameters is None:

query = "INSERT INTO hexagons (created\_at, hex\_id, hex\_resolution, hex\_color) VALUES (NOW(), %s, %s, ROW(%s, %s, %s)::color)"

self.\_\_insert\_query(

query,

hex\_id,

h3.get\_resolution(hex\_id),

r, g, b

)

return

query = "INSERT INTO hexagons (created\_at, hex\_id, hex\_resolution, hex\_color, avg\_water\_parameters) VALUES (NOW(), %s, %s, ROW(%s, %s, %s)::color, ROW(%s, %s, %s, %s, %s)::water\_parameters)"

self.\_\_insert\_query(

query,

hex\_id,

h3.get\_resolution(hex\_id),

r, g, b,

\_get\_tuple(water\_parameters.smell),

\_get\_tuple(water\_parameters.taste),

\_get\_tuple(water\_parameters.color),

\_get\_tuple(water\_parameters.muddiness),

\_get\_tuple(water\_parameters.general\_mineralization),

)

def insert\_address\_info(

self,

address: str,

coordinates: Point,

water\_parameters: Optional[WaterParameters]

):

if water\_parameters:

query = "INSERT INTO address\_info (created\_at, address, coordinates, water\_parameters) VALUES (NOW(), %s, ROW(%s, %s)::geo\_point, ROW(%s, %s, %s, %s, %s)::water\_parameters)"

self.\_\_insert\_query(

query,

address,

coordinates.latitude,

coordinates.longitude,

\_get\_tuple(water\_parameters.smell),

\_get\_tuple(water\_parameters.taste),

\_get\_tuple(water\_parameters.color),

\_get\_tuple(water\_parameters.muddiness),

\_get\_tuple(water\_parameters.general\_mineralization),

)

else:

query = "INSERT INTO address\_info (created\_at, address, coordinates) VALUES (NOW(), %s, ROW(%s, %s)::geo\_point)"

self.\_\_insert\_query(

query,

address,

coordinates.latitude,

coordinates.longitude,

)

def \_parse\_tuple(s: str) -> tuple[int, int, int]:

x, y, z = s.strip("()").split(",")

return int(x), int(y), int(z)

def \_parse\_parameter(name, units, value, max\_allowed\_concentration):

return Parameter(name, units, float(value), float(max\_allowed\_concentration))

def \_parse\_water\_params(db\_water\_params) -> WaterParameters | None:

if db\_water\_params is None:

return None

return WaterParameters(

smell=\_parse\_parameter(\*db\_water\_params[0]),

taste=\_parse\_parameter(\*db\_water\_params[1]),

color=\_parse\_parameter(\*db\_water\_params[2]),

muddiness=\_parse\_parameter(\*db\_water\_params[3]),

general\_mineralization=\_parse\_parameter(\*db\_water\_params[4]),

)

def \_parse\_hexagon(db\_row) -> Hexagon:

created\_at, hex\_id, hex\_resolution, hex\_color, water\_params = db\_row

return Hexagon(

created\_at=created\_at,

hex\_id=hex\_id,

hex\_resolution=hex\_resolution,

hex\_color=\_parse\_tuple(hex\_color),

avg\_water\_parameters=\_parse\_water\_params(water\_params)

)

def \_parse\_address\_info(db\_row) -> AddressInfo:

created\_at, address, coordinates, water\_params = db\_row

return AddressInfo(

created\_at=created\_at,

address=address,

coordinates=Point(latitude=float(coordinates[0]), longitude=float(coordinates[1])),

water\_parameters=\_parse\_water\_params(water\_params)

)

def \_get\_tuple(param: Parameter):

return param.name, param.units, param.value, param.max\_allowed\_concentration

V0\_create\_user.sql

CREATE USER water\_heatmap\_admin WITH PASSWORD '<password>' CREATEDB CREATEROLE;

V1\_create\_db.sql

CREATE DATABASE water\_heatmap\_db OWNER water\_heatmap\_admin;

V2\_create\_table.sql

CREATE TYPE PARAMETER as (

name TEXT,

units TEXT ,

value NUMERIC,

max\_allowed\_concentration NUMERIC

);

CREATE TYPE WATER\_PARAMETERS as (

smell PARAMETER,

taste PARAMETER,

color PARAMETER,

muddiness PARAMETER,

general\_mineralization PARAMETER

);

CREATE TYPE COLOR as (

red SMALLINT,

green SMALLINT,

blue SMALLINT

);

CREATE TABLE IF NOT EXISTS HEXAGONS (

id bigint GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

created\_at timestamp NOT NULL,

hex\_id TEXT NOT NULL,

hex\_resolution INTEGER NOT NULL,

hex\_color TEXT NOT NULL,

avg\_water\_parameters WATER\_PARAMETERS

);

INSERT INTO HEXAGONS VALUES (

NOW(),

'871f4e143ffffff',

7, -- resolution

ROW(

148,

211,

31

)::COLOR,

ROW(

ROW('Запах', 'баллы', 1.5, 2.0), -- smell

ROW('Привкус', 'баллы', 1.2, 2.0), -- taste

ROW('Цветность', 'градусы', 0.8, 1.5), -- color

ROW('Мутность', 'мг/дм3', 0.5, 1.0), -- muddiness

ROW('Общая минерализация \*\*\*', 'мг/дм3', 150, 200) -- general\_mineralization

)::WATER\_PARAMETERS

);

CREATE TYPE GEO\_POINT as (

latitude NUMERIC,

longitude NUMERIC

);

CREATE TABLE IF NOT EXISTS ADDRESS\_INFO (

id bigint GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

created\_at timestamp NOT NULL,

address TEXT NOT NULL,

coordinates GEO\_POINT NOT NULL,

water\_parameters WATER\_PARAMETERS

);

INSERT INTO ADDRESS\_INFO VALUES (

NOW(),

'Республика Беларусь, г.Минск, Газеты Звязда просп., 42',

ROW(

53.86037,

27.46016

)::GEO\_POINT,

ROW(

ROW('Запах', 'баллы', 0.25, 2.0), -- smell

ROW('Привкус', 'баллы', 1.2, 2.0), -- taste

ROW('Цветность', 'градусы', 0.8, 1.5), -- color

ROW('Мутность', 'мг/дм3', 0.5, 1.0), -- muddiness

ROW('Общая минерализация \*\*\*', 'мг/дм3', 150, 200) -- general\_mineralization

)::WATER\_PARAMETERS

);

# Б.4 ПОЛУЧЕНИЕ АДРЕСОВ ЖИЛЫХ ДОМОВ МИНСКА

import logging

import requests

from tqdm import tqdm

from bs4 import BeautifulSoup

from resources.utils import get\_path\_for\_saving

def get\_streets\_for\_letter(letter\_num, session):

"""Получаем список всех улиц для указанной буквы"""

url = f"https://ato.by/streets/letter/{letter\_num}"

try:

response = session.get(url)

response.raise\_for\_status()

soup = BeautifulSoup(response.text, 'html.parser')

street\_links = soup.select('div.intro ul li a[href^="/street/"]')

if not street\_links:

return []

streets = []

for link in street\_links:

street\_name = link.get\_text(strip=True)

street\_num = link['href'].split('/')[-1] # Извлекаем номер улицы из URL

streets.append({

'name': street\_name,

'num': street\_num

})

return streets

except Exception as e:

logging.error(f"Ошибка при получении улиц для буквы {letter\_num}: {e}")

return []

def get\_houses\_for\_street(street\_num, session):

"""Получаем все дома для указанной улицы"""

url = f"https://ato.by/street/{street\_num}"

try:

response = session.get(url)

response.raise\_for\_status()

soup = BeautifulSoup(response.text, 'html.parser')

house\_select = soup.find('select', {'id': 'hSHouseId'})

if not house\_select:

return []

houses = []

for option in house\_select.find\_all('option'):

if option['value'] != '0': # Пропускаем первый option

house\_number = option.get\_text(strip=True)

houses.append(house\_number)

return houses

except Exception as e:

logging.error(f"Ошибка при получении домов для улицы {street\_num}: {e}")

return []

def get\_all\_addresses():

"""Получаем все адреса Минска"""

addresses = []

geo\_prefix = 'Республика Беларусь, г. Минск'

with open(f'{get\_path\_for\_saving()}/minsk\_addresses.txt') as f:

lines = f.read().splitlines()

if lines:

return [f'{geo\_prefix}, {address}' for address in lines]

with requests.Session() as session:

logging.info(f'ATO.BY client. Starting web-scraping.')

session.headers.update({

'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'

})

for letter\_num in tqdm(range(1, 33)):

# logging.info(f"Processing letter #{letter\_num}")

streets = get\_streets\_for\_letter(letter\_num, session)

if not streets:

continue

for street in streets:

houses = get\_houses\_for\_street(street['num'], session)

for house in houses:

full\_address = f"{street['name']}, {house}"

addresses.append(f'{geo\_prefix}, {full\_address}')

return addresses

# Б.5 ПОЛУЧЕНИЕ ПОКАЗАТЕЛЕЙ ВОДЫ

"""

Клиент для сайта минскоговодоканала.

Предоставляем возможность получить данные о качестве воды с их сайта.

Поле \_\_headers - имитирует запрос из браузера

ВАЖНО! для того чтобы драйвер завелся успешно - необходимо, чтобы путь до него был в переменной $PATH

"""

import time

import selenium

from model.water\_parameters import Parameter, WaterParameters

from typing import Optional

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.common.keys import Keys

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.chrome.options import Options

import logging

def parse\_float(text: str) -> float:

if "/" in text:

text = text.split("/")[0].strip()

if "-" in text:

text = text.split("-")[1].strip() # берем максимум

if "<" in text:

text = text[1:]

if text == '':

text = '0.0'

return float(text.replace(",", "."))

def parse\_smell(td):

return Parameter(

name=td[0].find\_elements(By.TAG\_NAME, "span")[0].text,

units=td[1].text,

value=parse\_float(td[2].text), # has format 0/0

max\_allowed\_concentration=parse\_float(td[3].text),

)

def parse\_taste(td):

return Parameter(

name=td[0].find\_elements(By.TAG\_NAME, "span")[0].text,

units=td[1].text,

value=parse\_float(td[2].text),

max\_allowed\_concentration=parse\_float(td[3].text),

)

def parse\_color(td):

return Parameter(

name=td[0].find\_elements(By.TAG\_NAME, "span")[0].text,

units=td[1].text,

value=parse\_float(td[2].text),

max\_allowed\_concentration=parse\_float(td[3].text),

)

def parse\_muddiness(td):

return Parameter(

name=td[0].find\_elements(By.TAG\_NAME, "span")[0].text,

units=td[1].text,

value=parse\_float(td[2].text),

max\_allowed\_concentration=parse\_float(td[3].text),

)

def parse\_general\_mineralization(td):

return Parameter(

name=td[0].find\_elements(By.TAG\_NAME, "span")[0].text,

units=td[1].text,

value=parse\_float(td[2].text),

max\_allowed\_concentration=parse\_float(td[3].text),

)

class MinskVodokanalClient:

\_\_url: str = "https://minskvodokanal.by/water/home/"

"""

Создаем веб-сессию через библиотеку selenium.

Я использую Chrome, так как это самый популярный браузер

(соответственно для него веб-драйвер найти легче всего)

"""

def \_\_init\_\_(self):

chrome\_options = Options()

chrome\_options.add\_argument("--headless=new")

logging.debug("Starting webdriver for minskvodokanal")

self.\_\_driver = webdriver.Chrome(options=chrome\_options)

self.\_\_retries\_count = 2

logging.debug("Webdriver started")

def \_\_del\_\_(self):

self.\_\_driver.quit()

def v1\_request(self, address: str) -> Optional[WaterParameters]:

for i in range(self.\_\_retries\_count):

try:

logging.debug(f"MinskVodokanal GET request for {address}")

self.\_\_driver.get(self.\_\_url)

logging.debug("Send address to site")

input\_field = self.\_\_driver .find\_element(By.ID, "address-selector")

input\_field.send\_keys(address)

input\_field.send\_keys(Keys.RETURN) # RETURN == ENTER

logging.debug("Searching for info-params-0")

table\_name = "info-params-0"

wait = WebDriverWait(self.\_\_driver , 10) # Wait up to 10 seconds

wait.until(EC.presence\_of\_element\_located((By.ID, table\_name)))

time.sleep(0.5)

info\_params = self.\_\_driver.find\_elements(By.ID, table\_name)[0]

rows = info\_params.find\_elements(By.TAG\_NAME, "tr")

if len(rows) == 0:

continue

logging.debug("Start parsing into WaterParameters")

wp = WaterParameters(

smell=parse\_smell(rows[0].find\_elements(By.TAG\_NAME, "td")),

taste=parse\_taste(rows[1].find\_elements(By.TAG\_NAME, "td")),

color=parse\_color(rows[2].find\_elements(By.TAG\_NAME, "td")),

muddiness=parse\_muddiness(rows[3].find\_elements(By.TAG\_NAME, "td")),

general\_mineralization=parse\_general\_mineralization(

rows[4].find\_elements(By.TAG\_NAME, "td")

),

)

return wp

except selenium.common.exceptions.StaleElementReferenceException as e:

logging.error(e)

return None

# Б.5 ИСПОЛЬЗУЕМЫЕ МОДЕЛИ ДАННЫХ

from dataclasses import dataclass, is\_dataclass, asdict

from datetime import datetime

from json import JSONEncoder, JSONDecoder

from typing import Optional, Any, Dict

from model.water\_parameters import WaterParameters, Parameter

@dataclass

class Point:

latitude: float

longitude: float

class GeoPointEncoder(JSONEncoder):

def default(self, o):

return o.\_\_dict\_\_

@dataclass

class Hexagon:

created\_at: datetime

hex\_id: str

hex\_resolution: int

hex\_color: tuple[int, int, int]

avg\_water\_parameters: Optional[WaterParameters]

@dataclass

class AddressInfo:

created\_at: datetime

address: str

coordinates: Point

water\_parameters: Optional[WaterParameters]

class GeoEncoder(JSONEncoder):

def default(self, obj: Any) -> Any:

if isinstance(obj, datetime):

# формат ПГ

return obj.strftime('%Y-%m-%d %H:%M:%S.%f')

if is\_dataclass(obj):

result = asdict(obj)

if hasattr(obj, 'norm'):

result['norm'] = obj.norm()

return result

return super().default(obj)

class GeoDecoder(JSONDecoder):

def default(dct: Dict[str, Any]) -> Any:

if all(k in dct for k in ('latitude', 'longitude')):

return Point(\*\*dct)

if all(k in dct for k in ('name', 'units', 'value', 'max\_allowed\_concentration')):

return Parameter(\*\*{k: v for k, v in dct.items()

if k in ('name', 'units', 'value', 'max\_allowed\_concentration')})

if all(k in dct for k in ('smell', 'taste', 'color', 'muddiness', 'general\_mineralization')):

return WaterParameters(\*\*dct)

if 'created\_at' in dct and 'address' in dct and 'coordinates' in dct:

addr = AddressInfo(\*\*dct)

addr.created\_at = datetime.strptime(addr.created\_at, '%Y-%m-%d %H:%M:%S.%f')

return addr

if 'address' in dct and 'coordinates' in dct:

return AddressInfo(created\_at=datetime.now(), water\_parameters=None, \*\*dct)

return dct

def make\_hex\_id(h3\_res: int):

return f'hex\_res\_{h3\_res}\_id'

@dataclass

class Parameter:

name: str

units: str # units represented in russian language

value: float

max\_allowed\_concentration: float

def norm(self):

return self.value / self.max\_allowed\_concentration

@dataclass

class WaterParameters:

smell: Parameter

taste: Parameter

color: Parameter

muddiness: Parameter # мутность

general\_mineralization: Parameter

import os

def get\_path\_for\_saving() -> str:

dir\_path = os.path.dirname(os.path.realpath(\_\_file\_\_))

return dir\_path

# Б.6 РЕГИСТРАЦИЯ ОБРАБОТЧИКОВ

import logging

import h3

import model

import json

import schedule

from flask import Flask, jsonify

from external.pg.client import PgClient

from external.web.ato.client import get\_all\_addresses

from logic.cron.calculate\_water\_parameters\_task import calculate\_water\_parameters\_task, save\_coordinates, \

save\_aggregated\_hexagons\_information

from logic.geo.houses import dump\_addresses\_to\_file, read\_already\_fetched\_houses, enrich\_with\_hexagons, \

retrieve\_addresses\_info

from model.geo import make\_hex\_id, AddressInfo, Point, Hexagon

from model.water\_parameters import Parameter, WaterParameters

from external.pg.client import PgClient

from logic.geo.houses import retrieve\_address\_info

from logic.water\_quality.water\_parameters import retrieve\_water\_parameters

from model.geo import Hexagon, GeoEncoder

logging.basicConfig(level=logging.INFO)

def test\_enrich\_and\_print():

def enrich\_and\_print(addresses\_and\_coordinates, hex\_res):

enriched = enrich\_with\_hexagons(addresses\_and\_coordinates, hex\_res)

unique\_hex\_ids = {obj[make\_hex\_id(hex\_res)] for obj in enriched}

print(f'\nUnique hexes with hex\_res = {hex\_res} (len = {len(unique\_hex\_ids)})')

for hex\_id in unique\_hex\_ids:

print(f'\t{hex\_id}; hex\_childrens={h3.cell\_to\_children(hex\_id)}')

houses = read\_already\_fetched\_houses()

print(houses)

print( f'Total houses = {len(houses)}')

enrich\_and\_print(houses, 6)

enrich\_and\_print(houses, 7)

enrich\_and\_print(houses, 8)

enrich\_and\_print(houses, 9)

def enrich\_and\_save():

addresses = get\_all\_addresses()

houses = retrieve\_addresses\_info(addresses, geocoder\_requests\_limit=5)

dump\_addresses\_to\_file(houses)

def test\_pg\_client\_select\_address\_info():

pg\_client = PgClient()

target = 'Республика Беларусь, г.Минск, Газеты Звязда просп., 42'

address\_info: AddressInfo = pg\_client.get\_address\_info(target)

assert address\_info.address == target

assert address\_info.coordinates == Point(latitude=53.86037, longitude=27.46016)

assert address\_info.water\_parameters.smell == Parameter('Запах', 'баллы', 1.5, 2.0)

assert address\_info.water\_parameters.taste == Parameter('Привкус', 'баллы', 1.2, 2.0)

assert address\_info.water\_parameters.color == Parameter('Цветность', 'градусы', 0.8, 1.5)

assert address\_info.water\_parameters.muddiness == Parameter('Мутность', 'мг/дм3', 0.5, 1.0)

assert address\_info.water\_parameters.general\_mineralization == Parameter('Общая минерализация \*\*\*', 'мг/дм3', 150, 200)

def test\_pg\_client\_select\_hex\_info():

pg\_client = PgClient()

target = '871f4e143ffffff'

hexagon: Hexagon = pg\_client.get\_info\_about\_hex(target)

assert hexagon.hex\_id == target

assert hexagon.hex\_resolution == h3.get\_resolution(target)

assert hexagon.hex\_color == (148,211,31)

assert hexagon.avg\_water\_parameters.smell == Parameter('Запах', 'баллы', 1.5, 2.0)

assert hexagon.avg\_water\_parameters.taste == Parameter('Привкус', 'баллы', 1.2, 2.0)

assert hexagon.avg\_water\_parameters.color == Parameter('Цветность', 'градусы', 0.8, 1.5)

assert hexagon.avg\_water\_parameters.muddiness == Parameter('Мутность', 'мг/дм3', 0.5, 1.0)

assert hexagon.avg\_water\_parameters.general\_mineralization == Parameter('Общая минерализация \*\*\*', 'мг/дм3', 150, 200)

def test\_pg\_client\_select\_all\_hexes():

pg\_client = PgClient()

hexagons: list[Hexagon] = pg\_client.get\_all\_hexes\_with\_res(7)

hexagon = hexagons[0]

target = '871f4e143ffffff'

assert hexagon.hex\_id == target

assert hexagon.hex\_resolution == h3.get\_resolution(target)

assert hexagon.hex\_color == (148,211,31)

assert hexagon.avg\_water\_parameters.smell == Parameter('Запах', 'баллы', 1.5, 2.0)

assert hexagon.avg\_water\_parameters.taste == Parameter('Привкус', 'баллы', 1.2, 2.0)

assert hexagon.avg\_water\_parameters.color == Parameter('Цветность', 'градусы', 0.8, 1.5)

assert hexagon.avg\_water\_parameters.muddiness == Parameter('Мутность', 'мг/дм3', 0.5, 1.0)

assert hexagon.avg\_water\_parameters.general\_mineralization == Parameter('Общая минерализация \*\*\*', 'мг/дм3', 150, 200)

hexagon = hexagons[1]

target = '871f4e14effffff'

assert hexagon.hex\_id == target

assert hexagon.hex\_resolution == h3.get\_resolution(target)

assert hexagon.hex\_color == (148,211,31)

def test\_pg\_client\_insert\_hex():

pg\_client = PgClient()

pg\_client.insert\_hexagon(

'8711a0d80ffffff',

(123, 123, 123),

WaterParameters(

smell=Parameter('Запах', 'баллы', 1.5, 2.0),

taste=Parameter('Привкус', 'баллы', 1.2, 2.0),

color=Parameter('Цветность', 'градусы', 0.8, 1.5),

muddiness=Parameter('Мутность', 'мг/дм3', 0.5, 1.0),

general\_mineralization=Parameter('Общая минерализация \*\*\*', 'мг/дм3', 150, 200),

))

def test\_pg\_client\_insert\_address\_info():

pg\_client = PgClient()

pg\_client.insert\_address\_info(

'Республика Беларусь, г. Минск, Краснозвездная ул., 14А/6',

Point(latitude=53.908092, longitude=27.588861),

None

)

def test\_pg\_client\_select\_all\_available\_address\_info():

pg\_client = PgClient()

res = pg\_client.get\_all\_address\_info()

assert len(res) == 12

app = Flask(\_\_name\_\_)

schedule.every().monday.at('03:00', 'Europe/Minsk').do(save\_addresses)

schedule.every().day.at('22:00', 'Europe/Minsk').do(save\_coordinates)

schedule.every().day.at('23.00', 'Europe/Minsk').do(save\_water\_parameters)

if \_\_name\_\_ == "\_\_main\_\_":

test\_pg\_client\_select\_address\_info()

test\_pg\_client\_select\_hex\_info()

test\_pg\_client\_select\_all\_hexes()

test\_pg\_client\_select\_all\_available\_address\_info()

@app.route("/v1/hexagon/all/<int:hex\_res>/colors", methods=["GET"])

def get\_hexagons\_with\_colors(hex\_res: int):

pg\_client = PgClient()

all\_hexagons: list[Hexagon] = pg\_client.get\_all\_hexes\_with\_res(hex\_res)

hexagons\_with\_colors: list[tuple[str, tuple[int, int, int]]] = [(hexagon.hex\_id, hexagon.hex\_color) for hexagon in all\_hexagons]

return hexagons\_with\_colors

@app.route("/v1/hexagon/all/<int:hex\_res>/info", methods=["GET"])

def get\_hexagons\_info(hex\_res: int):

pg\_client = PgClient()

all\_hexagons: list[Hexagon] = pg\_client.get\_all\_hexes\_with\_res(hex\_res)

return all\_hexagons

@app.route("/v1/hexagon/<hex\_id>/info", methods=["GET"])

def get\_hexagon\_info(hex\_id: str):

pg\_client = PgClient()

hexagon = pg\_client.get\_info\_about\_hex(hex\_id)

return json.dumps(hexagon, cls=GeoEncoder, ensure\_ascii=False)

@app.route("/v1/address/<address>/info", methods=["GET"])

def get\_address\_info(address: str):

address\_info = retrieve\_address\_info(address)

if address\_info:

if address\_info.water\_parameters is None:

address\_info.water\_parameters = retrieve\_water\_parameters([address\_info])[0]

pg\_client = PgClient()

pg\_client.insert\_address\_info(

address\_info.address,

address\_info.coordinates,

address\_info.water\_parameters

)

return json.dumps(address\_info, cls=GeoEncoder, ensure\_ascii=False)

# Б.7 ОБНОВЛЕНИЕ ГЕО ИНФОРМАЦИИ

import datetime, json, logging, copy, h3, Yandex-geocoder

from typing import Optional

from tqdm import tqdm

from tqdm.contrib.logging import logging\_redirect\_tqdm

import model

from external.pg.client import PgClient

from external.yandex.geocoder.client import GeocoderClient

from model.geo import GeoPointEncoder, make\_hex\_id, AddressInfo, GeoEncoder, Point, GeoDecoder

from resources.utils import get\_path\_for\_saving

def read\_already\_fetched\_houses\_from\_resources\_file() -> list[AddressInfo]:

with open(f'{get\_path\_for\_saving()}/houses\_geo\_info.json') as f:

try:

return json.load(f, object\_hook=GeoDecoder.default)

except json.decoder.JSONDecodeError as e:

logging.error(f"Not valid json file, file not exists or empty file. Error: {e}")

return []

def read\_already\_fetched\_houses\_from\_pg\_client(addresses: list[str]) -> list[AddressInfo]:

pg\_client = PgClient()

already\_fetched = []

all\_address\_info = pg\_client.get\_all\_address\_info()

map\_from\_address\_info\_to\_idx = {address\_info.address: idx for idx, address\_info in enumerate(all\_address\_info)}

with logging\_redirect\_tqdm():

for address in addresses:

if address in map\_from\_address\_info\_to\_idx:

address\_idx = map\_from\_address\_info\_to\_idx[address]

already\_fetched.append(all\_address\_info[address\_idx])

return already\_fetched

def read\_already\_fetched\_houses(addresses) -> list[AddressInfo]:

logging.debug("Starting reading prefetch from file")

from\_file = read\_already\_fetched\_houses\_from\_resources\_file()

logging.debug("Starting reading prefetch from pg")

from\_pg: list[AddressInfo] = read\_already\_fetched\_houses\_from\_pg\_client(addresses)

logging.debug("Combine two data into one")

uniq\_address\_from\_pg = {address\_info.address for address\_info in from\_pg}

for address\_info in from\_file:

if address\_info.address not in uniq\_address\_from\_pg:

from\_pg.append(address\_info)

return from\_pg

def get\_from\_geocoder(address: str, geocoder: GeocoderClient) -> AddressInfo | None:

try:

coordinates: model.geo.Point = geocoder.coordinates(address)

return AddressInfo(

created\_at=datetime.datetime.now(),

address=address,

coordinates=coordinates,

water\_parameters=None,

)

except yandex\_geocoder.exceptions.NothingFound as e:

logging.warning(f'Not found coordinates for address={address}. e={e}')

except yandex\_geocoder.exceptions.InvalidKey as e:

logging.error("Limit of requests reached. Consider changing api-key or increasing limits")

return None

def retrieve\_addresses\_info(addresses: list[str], geocoder\_requests\_limit=None):

geocoder = GeocoderClient()

houses: list[AddressInfo] = read\_already\_fetched\_houses(addresses)

already\_fetched\_addresses = {obj.address for obj in houses}

with logging\_redirect\_tqdm():

need\_to\_process\_address = list(filter(lambda x: x not in already\_fetched\_addresses, addresses))

need\_to\_process\_address = need\_to\_process\_address[:geocoder\_requests\_limit or 10]

for address in tqdm(need\_to\_process\_address, desc="Fetch coordinates from geocoder"):

address\_info = get\_from\_geocoder(address, geocoder)

if address\_info:

houses.append(address\_info)

# dump\_addresses\_to\_file(houses)

return houses

def retrieve\_address\_info(address: str) -> Optional[AddressInfo]:

already\_fetched\_houses = read\_already\_fetched\_houses([address])

already\_fetched\_addresses = {obj.address: idx for idx, obj in enumerate(already\_fetched\_houses)}

if address in already\_fetched\_addresses:

return already\_fetched\_houses[already\_fetched\_addresses[address]]

logging.info(f"Fetch address {address} from geocoder")

geocoder = GeocoderClient()

return get\_from\_geocoder(address, geocoder)

import colorsys

from model.water\_parameters import WaterParameters

# Нормализованные значения параметров (0.0 - 1.0)

# Берём максимальное отклонение среди всех параметров

# Ограничиваем максимальное отклонение 1.0

# Преобразуем отклонение в цвет:

# 0.0 (0%) - зелёный (120° в HSV), 1.0 (100%) - красный (0° в HSV)

# Конвертируем HSV в RGB

def determine\_color(water\_params: WaterParameters):

if water\_params is None:

return 184, 183, 174 # grey color

norm\_smell = water\_params.smell.norm()

norm\_mineral = water\_params.general\_mineralization.norm()

norm\_color = water\_params.color.norm()

norm\_taste = water\_params.taste.norm()

norm\_muddiness = water\_params.muddiness.norm()

max\_deviation = max(norm\_smell, norm\_mineral, norm\_color, norm\_taste, norm\_muddiness)

max\_deviation = min(max\_deviation, 1.0)

hue = (1.0 - max\_deviation) \* 120.0 / 360.0

# Фиксированные насыщенность и яркость

saturation = 0.9

value = 0.9

r, g, b = colorsys.hsv\_to\_rgb(hue, saturation, value)

r = int(r \* 255)

g = int(g \* 255)

b = int(b \* 255)

return r, g, b

# Ведомость дипломного проекта