

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего
образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Кафедра

вычислительной техники

«Вычислительная техника»

Специальность: 09.05.01 «Применение и эксплуатация
автоматизированных систем специального
назначения»
Специализация: «Эксплуатация вычислительных машин,
комплексов, систем и сетей специального
назначения»

ДИПЛОМНЫЙ ПРОЕКТ

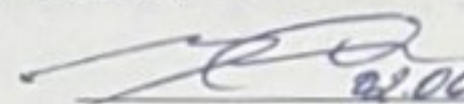
на тему

КЛИЕНТСКАЯ ЧАСТЬ ГЕОИНФОРМАЦИОННОЙ ПЛАТФОРМЫ
СБОРА ДАННЫХ С ИНТЕГРАЦИЕЙ МЕДИАКОНТЕНТА

Студент

 02.06.2025 Голышев Александр Андреевич
(подпись, дата) (ФИО полностью)

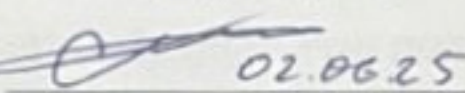
Руководитель

 02.06.2025 Зинкин С.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

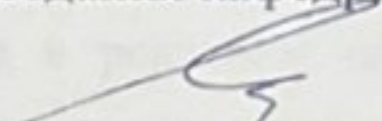
 02.06.2025 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Рецензент

Доцент Пенз ГТУ  02.06.25 Тетеревский С.В.
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 28.05.2025 №16)

Заведующий кафедрой

 Митрохин М.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично

(протокол заседания ГЭК от 19.06.2025. № 3)

Секретарь ГЭК

 Гурин Е.И.
(подпись) (фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
Митрохин М.А.
« 6 » 05 2025 г.

**ЗАДАНИЕ
НА ДИПЛОМНЫЙ ПРОЕКТ**

1. Студент Гольшев Александр Андреевич гр. 20ВОЭ1 факультета вычислительной техники.

2. Тема работы: «Клиентская часть геоинформационной платформы сбора данных с интеграцией медиаконтента».

Тема утверждена приказом ПГУ №601/о от "06" мая 2025

3. Руководитель работы д.т.н., профессор каф. ВТ «ПГУ» Зинкин С.А.

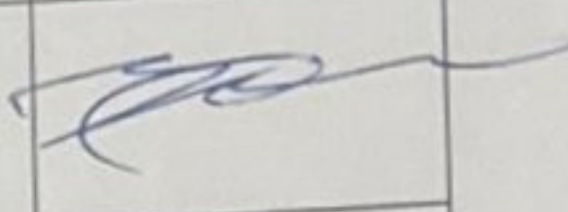
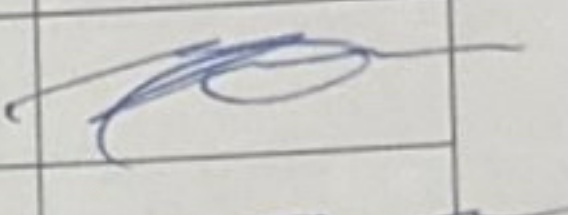
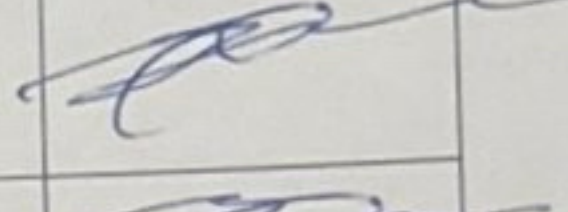
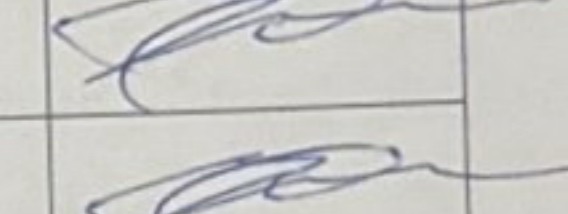
4. Задание на дипломную работу

Разработать клиентскую часть геоинформационной платформы для отображения и взаимодействия с физическими источниками данных. Система должна обеспечивать связь с сервером посредством REST API; иметь защищенную пользовательскую аутентификацию; осуществлять визуализацию статичных и подвижных объектов с маршрутом движения на заданной картографической подложке с возможностью просмотра фотоконтента (для статичных объектов) или потокового видео в реальном или архивном режиме (для статичных и подвижных объектов); предоставлять возможность фильтрации источников по выделенной географической области; обеспечивать производительное отображение большого количества источников на карте за счет кластеризации и оптимизированных запросов к серверу; отображать соответствующие наборам иконки формата png или svg на основе заданных параметров для них: ширина, высота, цвет, точка привязки к координатам; иметь в составе системы настроенный веб-сервер для удаленного доступа с контейнеризацией всего решения с использованием Docker.

5. Перечень подлежащих разработке вопросов.

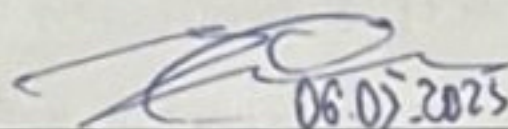
Требуется провести анализ предметной области и сформулировать требования к клиентской части системы, спроектировать архитектуру клиентского приложения, реализовать ключевые функциональные модули интерфейса и взаимодействие с сервером, провести тестирование функциональных модулей и оформить пояснительную записку.

6. Календарный график выполнения работы.

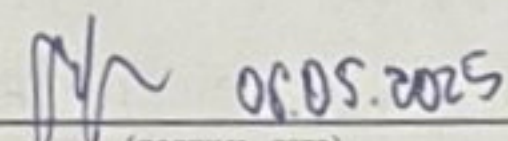
№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и формулирование требований к клиентской части системы	10%	06.05.2025 – 07.05.2025	
2	Проектирование архитектуры клиентского приложения	20%	08.05.2025 – 11.05.2025	
3	Реализация ключевых функциональных модулей интерфейса и взаимодействия с сервером	30%	12.05.2025 – 17.05.2025	
4	Тестирование функциональных модулей	20%	18.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 30.05.2025	

Дата выдачи задания «06» мая 2025 г.

Руководитель дипломного проекта

 06.05.2025 Зинкин С.А.
(подпись, дата) (фамилия, инициалы)

Задание к исполнению принял студент

 06.05.2025 Голышев А.А.
(подпись, дата) (фамилия, инициалы)

Работу к защите допустить
Декан факультета

 11.06.2025 Фионова Л.Р.
(подпись, дата) (фамилия, инициалы)

Реферат

Пояснительная записка содержит 130 листов, 24 рисунков, 12 таблиц, 9 источников и 3 приложения.

ГЕОИНФОРМАЦИОННАЯ ПЛАТФОРМА, КАРТОГРАФИЧЕСКИЙ ВЕБ-СЕРВИС, МЕДИАКОНТЕНТ, ПОТОКОВОЕ ВИДЕО, КЛАСТЕРИЗАЦИЯ ИСТОЧНИКОВ, REST API, КЛИЕНТСКОЕ ПРИЛОЖЕНИЕ

Объектом исследования является процесс отображения физических источников данных на картографической подложке и взаимодействие с информацией, которую они предоставляют, включая медиаконтент.

Целью работы является разработка клиентской части геоинформационной платформы сбора данных с интеграцией медиаконтента.

В процессе работы выполнены следующие задачи: анализ предметной области, определение функциональных требований к разрабатываемой системе, разработка ее архитектуры, выбор стека технологий, реализация механизмов отображения статичных и подвижных источников на карте, поддержка источников событийного и потокового типа, кастомизация отображаемых иконок, интеграция с серверной частью через REST API, а также выполнение контейнеризации приложения для последующего развертывания.

В результате проделанной работы разработано клиентской приложение в виде геоинформационной платформы, обеспечивающей отображение различных типов физических источников на карте, с возможностью получения предоставляемых ими данных, просмотра медиаконтента, просмотра маршрутов подвижных объектов, а также взаимодействие с источниками через различные модули пользовательского интерфейса.

					ПГУ1.090501.08.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Клиентская часть геоинформационной платформы сбора данных с интеграцией медиаконтента Пояснительная записка	Лит.		Лист	Листов
Разраб.		Голышев А.А.		02.06.25				4	130
Провер.		Зинкин С.А.		02.06.25					
Н. Контр.		Бычков А.С.		02.06.25					
Реченз.		Петренин С.		01.06.25					
Утверд.		Митрохин М.А.		02.06.25			ФВТ, каф. ВТ гр. 20ВОЭ1		

Содержание

Перечень сокращений и обозначений.....	8
Введение	9
1 Анализ предметной области	11
1.1 Описание предметной области	11
1.2 Анализ аналогов	12
1.3 Цели и задачи разрабатываемого приложения	14
1.4 Анализ целевой аудитории	15
2 Техническое задание	17
2.1 Назначение разработки	17
2.2 Требования к программе.....	17
2.2.1 Требования к функциональным характеристикам.....	17
2.2.2 Требования к надежности	18
2.2.3 Требования к составу и параметрам технических средств.....	18
2.3 Порядок контроля и приемки.....	19
3 Выбор инструментов разработки	21
3.1 Обоснование реализации клиентской части как веб-приложения	21
3.2 Обоснование выбора языка программирования.....	22
3.3 Обоснование выбора фреймворка	23
3.4 Используемые библиотеки и программные компоненты	23
4 Проектирование и архитектура системы.....	29
4.1 Общая архитектура системы	29
4.2 Структура компонентов	31
4.3 Инкапсуляция бизнес-логики.....	32
4.4 Взаимодействие с сервером	32
4.5 Глобальное хранилище данных	33
4.6 Модульность	33
5 Реализация функциональных блоков.....	35
5.1 Авторизация, аутентификация и защита маршрутов	35

5.1.1	Технология аутентификации JSON Web Token	35
5.1.2	Сервисный класс сетевых запросов авторизации.....	36
5.1.3	Централизованный HTTP-клиент	38
5.1.4	Локальная и внешняя авторизация Keycloak	40
5.1.5	Защита маршрутов.....	45
5.2	Обработка источников данных и графических обозначений.....	46
5.2.1	Получение информации о наборах источников с сервера	46
5.2.2	Локальное кеширование.....	47
5.2.3	Обработка иконок	51
5.3	Инициализация карты и картографических подложек	54
5.3.1	Конструирование слоев подложек.....	54
5.3.2	Инициализация карты	55
5.4	Отображение статичных источников	56
5.4.1	Слой статичных объектов	59
5.4.2	Клиентская кластеризация.....	61
5.4.3	Стилизация объектов на карте.....	64
5.5	Отображение подвижных источников.....	70
5.6	Интерфейс отображения событий, маршрутов и источников данных	72
5.7	Фильтрация по выделенной области	74
5.8	Медиаплееры	76
5.8.1	Фотоплеер.....	77
5.8.2	Видеоплеер	78
6	Развертывание и инфраструктура	83
6.1	Настройка веб-сервера Nginx.....	83
6.2	Контейнеризация приложения	83
7	Тестирование.....	85
8	Оценка затрат на разработку и внедрение.....	87
8.1	Организация процесса разработки	87
8.2	Расчет себестоимости программного обеспечения.....	89

8.3 Оценка эффективности разработки	93
8.4 Анализ рисков.....	94
9 Обеспечение безопасности труда при разработке	95
9.1 Актуальность вопросов безопасности	95
9.2 Техника безопасности при работе с электроприборами.....	95
9.3 Влияние ПЭВМ на физиологию человека	96
9.4 Техника безопасности при работе с ПК	97
Заключение	98
Список использованных источников	100
Приложение А.....	102
Приложение А.1 – Диаграмма вариантов использования	103
Приложение А.2 – Диаграмма развертывания	104
Приложение А.3 – Диаграмма последовательности	105
Приложение А.4 – Диаграмма деятельности.....	106
Приложение А.5 – Диаграмма компонентов.....	107
Приложение Б – Листинг программы	108
Приложение В – Презентация	124

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения:

ГИС – геоинформационная система, система для сбора, хранения, анализа и визуализации пространственных данных

API (Application Programming Interface) – интерфейс программного взаимодействия

Backend – серверная часть программного обеспечения, обрабатывающая бизнес-логику, работу с данными и взаимодействие с внешними системами

DOM (Document Object Model) – объектная модель документа, используемая для взаимодействия с HTML-элементами веб-страницы

Frontend – клиентская часть программного обеспечения, реализующая пользовательский интерфейс и взаимодействие с сервером через сетевые запросы

GPS (Global Positioning System) – система глобального позиционирования

HLS (HTTP Live Streaming) – технология потоковой передачи видео

IDC (International Data Corporation) – международная компания, предоставляющая аналитические данные по ИТ-рынку

JWT (JSON Web Token) – формат токенов для авторизации

REST API (Representational State Transfer API) – архитектурный стиль взаимодействия с сервером через HTTP

SPA (Single Page Application) – одностраничное веб-приложение, в котором обновление контента происходит без перезагрузки страницы

					ПГУ1.090501.08.001 ПЗ	Лист
						8
Из	Лист	№ докум.	Подпись	Дата		

Введение

В современном мире стремительно увеличиваются объемы хранимых данных, что стимулирует развитие соответствующих информационных технологий для их обработки и визуализации. Особенно актуальны задачи эффективной обработки геопространственных данных, в том числе мультимедийного характера. Согласно оценкам компании International Data Corporation, к 2025 году мировые объемы цифровых данных достигнут объема 175 зеттабайт [1]. Это обусловлено стремительным развитием городских информационных систем для государственных и муниципальных структур, а также модернизацией систем обеспечения городского мониторинга и безопасности.

В области обеспечения общественной безопасности и мониторинга городской инфраструктуры геоинформационные системы приобретают особую актуальность. Автоматизированные решения позволяют решать проблемы оперативной оценки обстановки, в результате чего риски информационных потерь снижаются, а эффективность принятий решений увеличивается. Такие системы позволяют осуществлять централизованную работу с различными физическими источниками данных, предоставляющие медиаконтент, в том числе в реальном времени [2]. Помимо этого, важна возможность функционирования таких решений в условиях, где отсутствует доступ к внешним сетям, что характерно для систем специального назначения. Автономность и способность работы в условиях замкнутой инфраструктуры без доступа к сети Интернет являются их необходимыми свойствами, особенно при использовании в закрытых корпоративных структурах. С точки зрения государственной политики Российской Федерации это обеспечивает технологический суверенитет и импортонезависимость программного обеспечения без интеграций зарубежных платформ и сервисов.

Разработка таких систем имеет значимость для городских и региональных администраций, служб безопасности, органов транспортной инфраструктуры, логистических компаний и других структур, деятельность которых связана с анализом геопространственных данных. Данного рода автоматизированные

					ПГУ1.090501.08.001 ПЗ	Лист
						9
Из	Лист	№ докум.	Подпись	Дата		

системы обеспечивают прозрачность процессов и управляемость данными, что значительно снижает нагрузку на персонал и позволяет минимизировать риски, связанные с человеческим фактором [3].

Целью настоящей дипломной работы является проектирование и разработка клиентской части геоинформационной платформы, предназначенной для отображения физических источников и визуальной оценки медиаданных, связанных с ними. Разрабатываемая система ориентирована на работу в условиях локальной инфраструктуры с учетом специфических требований к производительности.

					ПГУ1.090501.08.001 ПЗ	Лист
						10
Из	Лист	№ докум.	Подпись	Дата		

1 Анализ предметной области

1.1 Описание предметной области

В современных условиях наблюдается интенсивный рост объема геопространственных данных, поступающих от различных физических устройств. Обработка и визуализация такой информации являются одними из ключевых задач в области территориального мониторинга городов, районов и регионов. Для их решения необходимы информационные системы, объединяющие обработку пространственной информации и мультимедийных данных, а именно фото- и видеоматериалов, предоставляемых распределенными физическими источниками.

Геоинформационные платформы, совмещающие в себе картографию с медиаданными, имеют широкое применение в сфере городской логистики, транспортного мониторинга, обеспечения безопасности и экологического контроля. Ключевой задачей таких систем является возможность отображения большого количества физических источников с удобным и прозрачным доступом к предоставляемому ими медиаконтенту. Например, при мониторинге транспортных средств система позволяет отобразить местоположение патрульных машин с возможностью просмотра в реальном времени изображения с их видеорегистраторов.

Предметная область включает в себя механизмы сбора, агрегации, обработки и визуализации геопространственных данных от таких устройств как видеокамеры, камеры фотофиксации, GPS-трекеры. Особенностью таких систем является необходимость в визуализации как статичных, так и подвижных объектов. Статичные объекты требуют эффективного отображения с возможностью кластеризации и фильтрации, а подвижные – отображения траекторий маршрутов движения и текущего положения [4].

Актуальное значение имеет проблема избыточной плотности геопространственных данных. В результате этого происходит снижение удобства взаимодействия с пользовательским интерфейсом и усложнение визуального анализа. Для решения данной проблемы необходимо разработать

					ПГУ1.090501.08.001 ПЗ	Лист
						11
Из	Лист	№ докум.	Подпись	Дата		

оптимизированные алгоритмы визуализации таких случаев.

Критически важным требованием является обеспечение автономности работы системы. Для использования её в ведомственных структурах необходима возможность полного выполнения ключевого функционала в условиях замкнутой инфраструктуры без доступа к сети Интернет и внешним сервисам.

Таким образом, клиентская часть геоинформационной платформы должна быть ориентирована на:

- высокопроизводительную работу с большими наборами геопространственных данных;
- оптимизированную визуализацию физических источников сбора данных при их высокой плотности расположения;
- интегрированную поддержку просмотра мультимедийного контента;
- автономную эксплуатацию в условиях замкнутой инфраструктуры.

1.2 Анализ аналогов

Для определения требований к клиентской части геоинформационной платформы целесообразно рассмотреть существующие аналоги на российском рынке. При проведении анализа учитывалась функциональность пользовательского интерфейса, возможности визуализации данных, а также взаимодействие с геоданными и мультимедийной информацией, которую они могут предоставлять. Для рассмотрения аналогов были выбраны 4 самых зрелых и популярных систем, решающих задачи мониторинга геопространственных данных, а именно NextGIS, Geomixer (СКАНЭКС), GIS WebServer SE (КБ «Панорама») и зарубежная платформа Wialon, недавно ушедшая из России.

NextGIS предоставляет обширный функционал отображения геообъектов, включающий управление слоями данных, а также картографическими подложками. Поддержка мультимедийного контента ограничена только возможностью просмотра изображений, отображение видеопотока в данной системе не предусмотрено. Интерфейс приложения преимущественно адаптирован только для отображения статичных объектов, возможность для отображения маршрутов

					ПГУ1.090501.08.001 ПЗ	Лист
						12
Из	Лист	№ докум.	Подпись	Дата		

подвижных источников отсутствует. Система достаточно адаптирована для работы с большим количеством данных, а также способна функционировать в изолированной среде без доступа к сети Интернет [5].

Geomixer (СКАНЭКС) предоставляет похожий функционал с предыдущей системой. Система преимущественно адаптирована для работы со статичными источниками, а работа с медиаданными ограничивается только возможностью прикрепления фото к объектам [6].

GIS WebServer SE (КБ «Панорама») обладает схожим функционалом и предназначена для работы только со статичными источниками и их атрибутивной информацией. Работа с видео и потоковым медиаконтентом не предусмотрена и инструментов для отображения подвижных объектов в системе не предусмотрено [7].

Wialon в большей части адаптирован для отображения транспорта и предоставляет обширный функционал управления подвижными объектами, включая маршруты и отображение видеопотока. Пользователь может просматривать прямые трансляции, архивные видеофрагменты. Более того, можно установить собственную картографическую подложку. Интерфейс системы ориентирован для её использования в сфере логистики [8].

Сравнительная таблица аналогов представлена в таблице 1.

Таблица 1 – Сравнительная таблица аналогов

	NextGIS	Geomixer (СКАНЭКС)	GIS WebServer SE	Wialon
Управление слоями данных	Да	Да	Да	Частично
Отображение фото	Да	Да	Да	Да
Отображение видео	Нет	Нет	Нет	Да
Мониторинг статичных объектов	Да	Да	Да	Да
Мониторинг подвижных объектов	Нет	Нет	Нет	Да
Автономная работа	Да	Да	Да	Да

Проведенный анализ показывает, что существующие ГИС-платформы на российском рынке обладают базовым развитым функционалом для отображения

геоданных, но их возможности по отображению подвижных объектов и работе с видеоконтентом не предусмотрены. Разрабатываемое приложение нацелено на устранение данных ограничений и предлагает комплексный подход к мониторингу. Оно сочетает в себе отображение геоданных с полноценной встроенной поддержкой медиаконтента, визуализацию подвижных объектов с возможностью просмотра маршрутов, а также предусматривает соответствующие механизмы оптимизации при работе с большим объемом данных, позволяющие пользователю максимально эффективно просматривать и получать данные с необходимых ему источников. Более того, система способна к функционированию в условиях замкнутой инфраструктуры.

Данный функционал делает систему конкурентоспособной и актуальной с учетом современных инженерных потребностей.

1.3 Цели и задачи разрабатываемого приложения

На основе анализа предметной области и обзора существующих похожих решений сформулирована общая цель дипломной работы.

Целью дипломной работы является разработка клиентской части геоинформационной платформы сбора данных с интеграцией медиаконтента. Платформа ориентирована на отображение информации, поступающей от распределенных физических устройств (источников), включая просмотр событийных и потоковых данных.

Для достижения поставленной цели необходимо решить следующие задачи:

- реализовать отображение картографических подложек;
- реализовать отображение статичных и подвижных объектов на карте;
- реализовать просмотр событий для источников событийного типа;
- реализовать просмотр маршрутов для источников подвижного типа;
- реализовать просмотр фотоконтента, связанного с событиями;
- реализовать просмотр видеоконтента в архивном режиме и режиме прямой трансляции для источников потокового и подвижного типа;
- реализовать фильтрацию источников по выделенной области;

					ПГУ1.090501.08.001 ПЗ	Лист
						14
Из	Лист	№ докум.	Подпись	Дата		

- реализовать клиентскую кластеризацию статичных объектов с их оптимизированным отображением при высокой плотности расположения;
- обеспечить автономную работу пользовательского интерфейса без доступа к сети Интернет.

Диаграмма вариантов использования представлена на рисунке 1.



Рисунок 1 – Диаграмма вариантов использования

1.4 Анализ целевой аудитории

Целевой аудиторией разрабатываемой системы являются организации, осуществляющие мониторинг обстановки путем анализа событийной или потоковой информации, собранной с физических устройств сбора данных.

Информация может собираться либо со статичных источников, например,

фото- или видеокамер, или подвижных, например, беспилотные летательные аппараты. Пользователями системы могут быть операторы центров ситуационного мониторинга, диспетчеры в логистических или охранных организациях, специалисты силовых структур, обеспечивающих безопасность путем контроля городской или промышленной инфраструктуры.

Основными задачами, которые решают пользователи разрабатываемой системы, являются отслеживание физических устройств сбора данных, распределенных в пространстве, просмотр информации о них, оценка событий фотофиксации, анализ обстановки путем просмотра видеоконтента с камер за определенный промежуток времени или в режиме прямой трансляции если он доступен, а также фильтрация наборов источников по выделенной области на карте. Данный функционал обеспечивает высокую наглядность данных и эффективность принятия решений оператором системы [9].

Среди характерных требований пользователей системы из проанализированных отраслей может выделить следующие:

- простота и интуитивная понятность интерфейса;
- минимальное количество действий для выполнения ключевых функций;
- визуально оптимизированная работа системы при большой плотности геообъектов;
- возможность функционирования системы без доступа к сети Интернет.

Таким образом, на основе анализа потенциальных пользовательских сценариев, характера задач и ограничений эксплуатации была спроектирована архитектура и ключевая функциональность приложения.

					ПГУ1.090501.08.001 ПЗ	Лист
						16
Из	Лист	№ докум.	Подпись	Дата		

2 Техническое задание

2.1 Назначение разработки

Разрабатываемое программное обеспечение представляет собой клиентскую часть геоинформационной платформы. Назначением системы является отображение распределенных в пространстве физических источников сбора данных, включая отображение картографических подложек, визуализацию зафиксированных событий и работу со связанным видеоконтентом.

2.2 Требования к программе

2.2.1 Требования к функциональным характеристикам

Клиентская часть геоинформационной платформы должна обеспечивать следующие функциональные возможности, доступные пользователю через пользовательский интерфейс:

- обеспечивать авторизацию пользователя через локальный сервер или внешний провайдер Keycloak;
- обеспечивать защиту маршрутов сайта от неавторизованного доступа;
- обеспечивать отображение картографических подложек;
- обеспечивать загрузку и отображение физических источников данных (статичных и подвижных);
- отображать источники на карте в виде иконок с учетом их координат, заданной якорной точки привязки и принадлежности к набору;
- отображать видимые источники на карте в виде отдельного списка;
- обеспечивать фильтрацию источников по выделенной области на карте;
- обеспечивать просмотр зафиксированных событий для источников событийного типа;
- обеспечивать просмотр маршрутов для источников подвижного типа;
- обеспечивать воспроизведение видеопотоков от источников потокового и подвижного типов;
- поддерживать просмотр видеопотока в режиме прямой трансляции;

					ПГУ1.090501.08.001 ПЗ	Лист
						17
Из	Лист	№ докум.	Подпись	Дата		

- поддерживать просмотр видеопотока в архивном режиме за выбранный период времени;
- обеспечивать отображение маршрутов движения на карте для источников подвижного типа;
- обеспечивать воспроизведение видеозаписей, связанных с маршрутами;
- обеспечивать кластеризацию объектов и их визуально оптимизированное отображение на карте при высокой плотности источников;
- обеспечивать полноценную автономную работу интерфейса без подключения к сети Интернет.

2.2.2 Требования к надежности

Клиентское приложение должно обеспечивать стабильную и бесперебойную работу в условиях обработки большого объема данных. В случае нестабильного соединения с сервером интерфейс должен оставаться доступным. Все сетевые ошибки должны корректно обрабатываться с выдачей соответствующего уведомления пользователю. При работе в условиях недоступности сети Интернет пользовательский интерфейс должен сохранять свою работоспособность и обеспечивать взаимодействие со всем ключевым функционалом. Передача данных между клиентом и сервером должна осуществляться по защищенному протоколу HTTPS.

2.2.3 Требования к составу и параметрам технических средств

Клиентская часть системы должна обеспечивать работу на пользовательских ПК с поддержкой веб-браузеров. Минимальные требования для клиентской части:

- операционная система: Windows 7 и выше, macOS 10.12 и выше, или актуальные дистрибутивы Linux (например, Ubuntu 18.04 и выше);
- процессор: Intel Core i3 с тактовой частотой не менее 2.0 ГГц или эквивалентный;
- оперативная память: не менее 4 ГБ;
- свободное место на диске: не менее 2 ГБ;
- дисплей: диагональ не менее 14 дюймов, разрешение не менее 1280x720 пикселей;

					ПГУ1.090501.08.001 ПЗ	Лист
						18
Из	Лист	№ докум.	Подпись	Дата		

- браузер: Chrome 90+, Firefox 90+, Edge 90+, Safari 14+;
- устройства ввода: клавиатура и мышь.

Серверная часть, на котором развернуто клиентское приложение, должна обеспечивать хранение данных, обработку запросов и выполнение бизнес-логики.

Минимальные требования для серверной части:

- операционная система: Windows Server 2016 и выше, Ubuntu Server 18.04 и выше или аналогичная серверная ОС;
- процессор: Intel Xeon или эквивалентный с частотой не менее 2.5 ГГц;
- оперативная память: не менее 8 ГБ;
- свободное место на диске: не менее 20 ГБ;
- дисплей: диагональ не менее 19 дюймов;
- устройства ввода: клавиатура и мышь.

Для обеспечения масштабируемости и разворачиваемости клиентского приложения должны соблюдаться следующие инфраструктурные требования:

- приложение должно быть развернуто в контейнеризованной среде с использованием Docker;
- взаимодействие между клиентской и серверной частями должно осуществляться по протоколу HTTPS с использованием SSL/TLS-сертификатов.

2.3 Порядок контроля и приемки

Контроль корректной реализации проводится путем функционального тестирования. Система соответствует техническому заданию при успешном выполнении следующих требований:

- весь функционал, описанный в техническом задании, реализован и корректно работает;
- система демонстрирует устойчивую работу при различных сценариях использования и обрабатываемых объемах данных;
- ошибки в работе сервера или клиента не приводят к сбоям всего клиентского приложения;
- приложение сопровождается документацией, включающей

					ПГУ1.090501.08.001 ПЗ	Лист
						19
Из	Лист	№ докум.	Подпись	Дата		

пояснительную записку, листинг программы и демонстрационные результаты.

Приемка осуществляется в присутствии заказчика на основании успешного прохождения серии тестов и выполнения всех заявленных функциональных требований.

					ПГУ1.090501.08.001 ПЗ	Лист
						20
Из	Лист	№ докум.	Подпись	Дата		

3 Выбор инструментов разработки

3.1 Обоснование реализации клиентской части как веб-приложения

В рамках данной дипломной работы клиентская часть реализована в виде одностраничного веб-приложения, что обосновано как технологическими, так и функциональными требованиями к системе. Данный подход имеет название Single Page Application (SPA).

Реализованное приложение данного типа позволяет обеспечить более высокую отзывчивость интерфейса, избежать перезагрузок страницы при необходимости обновления динамических данных, а также снизить сетевую нагрузку за счет загрузки только необходимых для отображения данных.

В качестве альтернатив могли быть рассмотрены:

- десктопное приложение (на базе Electron, Qt, .NET и др.);
- мобильное приложение (например, с использованием Flutter или React Native);
- классическое многостраничное веб-приложение (MPA).

Однако каждый из альтернативных подходов имеет ограничения:

- десктопное приложение потребовало бы реализации и поддержки отдельных сборок под Windows, macOS и Linux, что увеличивает трудозатраты и снижает гибкость развёртывания, особенно в среде с ограниченным доступом к системным ресурсам;
- мобильные приложения не соответствуют основному сценарию использования, а именно работе оператора на стационарной рабочей станции с полноразмерной картой, множеством интерфейсных элементов и возможностью одновременного взаимодействия с несколькими модулями (медиаплеерами, фильтрацией, слоями и т.п.);
- MPA (многостраничные приложения) уступают SPA по отзывчивости, требуют частых перезагрузок страниц и не подходят для реализации единого состояния интерфейса, что критично при работе с интерактивной картой.

Выбор использования веб-технологий с использованием подхода SPA

					ПГУ1.090501.08.001 ПЗ	Лист
						21
Из	Лист	№ докум.	Подпись	Дата		

обоснован тем, что разработанная клиентская часть приобретает свойство кроссплатформенности. При таком подходе приложение независимо от платформы, будь то Windows, macOS или Linux, так как работа с ним осуществляется непосредственно в браузере. Разработка десктопного приложения не предоставила бы такой гибкости, так как требовала разных сборок под различные операционные системы.

3.2 Обоснование выбора языка программирования

В качестве языка программирования для разработки клиентской части веб-приложения был выбран JavaScript. Он поддерживается всеми современными браузерами, позволяет создавать полнофункциональные интерактивные пользовательские интерфейсы, а также не требует установки дополнительного программного обеспечения.

Язык является мультипарадигменным и поддерживает как функциональный, так и объектно-ориентированный стиль программирования. Он имеет большую экосистему различных библиотек и фреймворков, что ускоряет разработку и упрощает поддержку кода.

В качестве альтернативы был рассмотрен WebAssembly с языками Rust и C++. При использовании данного подхода можно увеличить производительность работы клиентской части, однако для манипуляции с DOM-деревом требуется связка с JavaScript, а в контексте работы с картографией значительно возрастает сложность реализации. Учитывая, что основной задачей является визуализация геоданных и медиаконтента, а также техническое задание предполагает разработку оптимизированных алгоритмов работы с большим объемом данных, использование WebAssembly является нецелесообразным.

Таким образом, использование JavaScript является обоснованным выбором с точки зрения производительности и стека используемых технологий для разработки.

					ПГУ1.090501.08.001 ПЗ	Лист
						22
Из	Лист	№ докум.	Подпись	Дата		

3.3 Обоснование выбора фреймворка

Для разработки веб-приложений на языке JavaScript особой популярностью пользуются различные фреймворки. Одними из самых используемых являются React, Angular и Vue. Все три фреймворка строятся вокруг компонентов - изолированных и переиспользуемых модулей интерфейса. Все решения используют реактивную модель обновления данных, то есть интерфейс автоматически обновляется при изменении данных. Фреймворки используют подход Single Page Application, при которой приложение загружается один раз, а затем динамически обновляет содержимое.

У каждого из них имеется развитая экосистема, включая средства для маршрутизации, управления состоянием, сборки и разработки. Это позволяет выстраивать полноценный цикл разработки. Если рассматривать фреймворки с точки зрения подхода к разработке React хорошо проявляет себя когда нужен полный контроль, гибкая архитектура, активное масштабирование; использование Vue обосновано когда важна стабильность и скорость разработки, то есть имеется базовая структура, но многое можно адаптировать под себя; Angular же, в свою очередь, выигрывает при работе в больших командах разработчиков и корпоративных системах, где нужна строгая структура, модульность и управляемость. Для построения клиентской части был выбран Vue, так как он предлагает сбалансированный подход.

В нём сочетаются высокая производительность, гибкость, чистота архитектуры и богатая экосистема.

3.4 Используемые библиотеки и программные компоненты

Для реализации клиентской части геоинформационной платформы был проанализирован и использован ряд библиотек и инструментов, предоставляющие различные функциональные возможности.

Выбор каждой из них основан на ключевых требованиях к функционалу системы, включая отображение картографической информации и просмотр

					ПГУ1.090501.08.001 ПЗ	Лист
						23
Из	Лист	№ докум.	Подпись	Дата		

видеокарты.

Одним из ключевых решений является выбор картографического провайдера, так как именно данный элемент интерфейса занимает центральную роль в реализации функционала системы. Существует большое множество различных решений, но не все они подходят для реализации задач, которые требуют полной автономной работы приложения без доступа к сети интернет и в условиях собственной инфраструктуры. Ниже рассмотрены самые популярные библиотеки, а также проанализированы причины, по которым их использование может быть нецелесообразным.

Google Maps API - предоставляет возможность для загрузки и просмотра объектов на карте. Возможность загрузить собственную картографическую подложку отсутствует, можно только модифицировать существующую. Функционал поддерживает отрисовку как стандартных геометрий типа точек и линий, так и сложных, типа полигонов. Так как каждый объект на карте является полноценным DOM-элементом, библиотека имеет плохую поддержку работы с большим количеством геоданных, даже с учетом встроенного механизма клиентской кластеризации. Помимо этого, решение не будет работать без доступа к интернету, а также не поддерживает развертывания в изолированном пространстве.

Yandex Maps API - по аналогии с предыдущим решением предоставляет возможность для загрузки и просмотра объектов на карте, подключение собственной картографической подложки отсутствует, пользование осуществляется только подложками непосредственно от Яндекса. Имеется поддержка как стандартных геометрий типа точек и линий, так и сложных, типа полигонов. Библиотека не способна функционировать без доступа к сети интернет, а также не предоставляет возможности развертывания в изолированном пространстве.

Leaflet - является более гибким решением по сравнению с предыдущими аналогами. В отличие от предыдущих библиотек позволяет подключить собственную картографическую подложку. Поддержка работы с большим объемом данных так же реализована в неполной степени, так как каждый объект на карте

					ПГУ1.090501.08.001 ПЗ	Лист
						24
Из	Лист	№ докум.	Подпись	Дата		

является полноценным DOM элементом. Решение поддерживает работу без доступа к сети интернет и позволяет вести работу в изолированном пространстве без какого-либо доступа к внешним ресурсам вне собственной инфраструктуры системы.

Mapbox GL - мощное решение для визуализации данных на картографической подложке. Решение не имеет проблем с отображением большого количества объектов поскольку использует WebGL рендеринг. Помимо этого, библиотека позволяет подключить собственную картографическую подложку, может работать без доступа к сети интернет и способна функционировать в изолированном окружении. Серьезным недостатком является то, что поскольку используется WebGL рендеринг отсутствует возможность динамического задания стилей непосредственно через JavaScript код. Это означает то, что стилизация объектов работает только на основе заранее заданных параметров.

OpenLayers является наилучшим решением. Решение позволяет подключать собственные картографические подложки по любым протоколам, гибко и на низком уровне управлять поведением геообъектов. Засчет Canvas/WebGL рендеринга наилучшим образом справляется с обработкой большого количества данных на карте по сравнению с предыдущими аналогами. Помимо этого, позволяет динамически управлять стилизацией объектов на карте, способна работать без доступа к сети интернет и полным образом функционировать в изолированной среде.

Наглядное сравнение аналогов картографических провайдеров представлено в таблице 2.

					ПГУ1.090501.08.001 ПЗ	Лист
						25
Из	Лист	№ докум.	Подпись	Дата		

Таблица 2 – Наглядное сравнение аналогов картографических провайдеров

Критерий	Google Maps	Yandex Maps	Leaflet	Mapbox GL	OpenLayers
Отображение геометрий типа точек	Да	Да	Да	Да	Да
Отображение геометрий типа линий и полигонов	Да	Да	Да	Да	Да
Подключение собственной картографической подложки	Нет	Нет	Да	Да	Да
Функционирование без доступа к сети интернет	Нет	Нет	Нет	Да	Да
Оптимизированная работа с большим количеством геоданных	Нет	Нет	Нет	Да	Да
Гибкость стилизации	Нет	Нет	Да	Нет	Да
Функционирование без внешних API	Нет	Нет	Да	Да	Да

Таким образом, проведенный анализ показал, что библиотека OpenLayers подходит наилучшим образом для реализации функционала приложения.

Решение обладает наибольшей гибкостью, предоставляет возможности управления визуализацией объектов на низком уровне и обладает необходимыми механизмами для оптимизированного отображения большого количества данных на картографической подложке.

Для ускорения разработки и соблюдения единого визуального стиля использование UI-фреймворка Vuetify является обоснованным решением. Он удобно интегрируется с Vue и предоставляет множество готовых элементов интерфейса – таблицы, карточки, формы, диалоговое окна и многое другое. Использование такого решения позволяет сосредоточиться на бизнес-логике приложения, а не на разработке базовых UI элементов. Кроме того, фреймворк позволяет гибко управлять цветовыми схемами приложения.

Для реализации маршрутизации, то есть переходами между страницами, был выбран Vue Router. Он является частью экосистемы Vue, разрабатывается и поддерживается теми же разработчиками фреймворка.

При помощи данного решения можно гибко контролировать доступ к тем или иным маршрутам, проверять права доступа, или отменять переход, если у пользователя нет доступа. Всё это делает его лучшим выбором для решения данной задачи.

Для централизованного управления состоянием приложения был выбран Vuex. Данное решение так же является частью экосистемы Vue. Оно решает проблему множества удаленных компонентов приложения, которые должны иметь доступ к одним и тем же данным. Таким образом не совершаются лишние операции их получения для каждого модуля отдельно.

В целях реализации дополнительного механизма аутентификации для системы управления идентификации и доступом Keycloak была выбрана библиотека keycloak-js. Она позволяет обеспечить интеграцию в приложение, поддерживает протоколы OAuth2 и OpenID Connect. Библиотека предоставляет возможность инициализации Keycloak-клиента, получения и обновления токенов, проверку ролей, управление сессией пользователя, редиректы на страницу входа/выхода. Более того, данное решение является официальным и поддерживаемым, гибко интегрируется с Vue.

Для взаимодействия приложения с серверной частью посредством HTTP-запросов была выбрана библиотека axios. Данное решение является лучшим выбором, так как поддерживает автоматическую сериализацию и десериализацию данных, позволяет устанавливать глобальные настройки для работы с сетевыми запросами, например, установить базовый адрес URL, заголовки по умолчанию, а также имеет механизм перехватчиков запросов и ответов. Данный функционал может активно использоваться для логирования, обработки ошибок, автообновления токенов и другого функционала. Более того, библиотека позволяет удобным образом контролировать ошибки и статусы при работе с HTTP-запросами.

					ПГУ1.090501.08.001 ПЗ	Лист
						27
Из	Лист	№ докум.	Подпись	Дата		

Для работы с видеопотоком библиотека video.js является одним из самых популярных и мощных решений. Она позволяет работать со множеством различных форматов видео, предоставляет возможность задавать собственную стилизацию видеоплееру, позволяет гибко настраивать интерфейс для воспроизведения видеоконтента как в режиме прямой трансляции, так и в режиме воспроизведения видеопериода. Данный функционал делает данную библиотеку отличным решением для использования в разработке приложения.

Для реализации подвижных и растягиваемых пользователем элементов интерфейса была выбрана библиотека vue-draggable-resizable, а для всплывающих уведомлений различного рода – vue-notification.

					ПГУ1.090501.08.001 ПЗ	Лист
						28
Из	Лист	№ докум.	Подпись	Дата		

4 Проектирование и архитектура системы

4.1 Общая архитектура системы

Для обеспечения масштабируемости и автономности инфраструктура система спроектирована по модульному принципу, где каждый элемент развернут в изолированной среде с определенным для него интерфейсом.

Клиентская часть, а именно frontend, представляет собой Docker-контейнер, состоящий из двух основных компонентов: веб-приложения и веб-сервера. Приложение, разработанное при помощи фреймворка Vue, реализует весь пользовательский интерфейс и сетевые запросы. В роли веб-сервера и обратного прокси-сервера для реализации взаимодействия браузера с системой по протоколу HTTPS выступает Nginx. Он обслуживает статические ресурсы, управляет маршрутизацией сетевых запросов к backend и сервису внешней аутентификации Keycloak.

Backend отвечает за реализацию серверной бизнес-логики. Для взаимодействия с клиентской частью он предоставляет определенный интерфейс REST API. Помимо этого, backend отвечает за обработку входящих данных от физических источников данных, например, камер фотофиксации или видеокамер, а также синхронизируется с внешним сервисом аутентификации Keycloak для обеспечения доступа к серверным ресурсам пользователей. Все взаимодействия клиентского веб-приложения с сервером осуществляется через веб-сервер Nginx с применением JWT-токенов.

Сервис аутентификации Keycloak является альтернативным способом авторизации пользователей. Он взаимодействует как с frontend, так и с backend. Клиентская часть синхронизируется с сервисом Keycloak для контроля аутентификации пользователя, получения токенов доступа, а также, при необходимости, перенаправление на страницу авторизации.

Синхронизация бэкенда заключается в цели валидации аутентификационных данных, полученных пользователем в процессе авторизации.

Панель администратора является отдельным веб-приложением,

					ПГУ1.090501.08.001 ПЗ	Лист
						29
Из	Лист	№ докум.	Подпись	Дата		

принимаящим участие в общей архитектуре, но не описанным в рамках данной дипломной темы. Она отвечает за настройку различных технических параметров всей системы, в том числе управление учетными записями пользователей, а также наборами источников различных типов, в которые группируются физические источники сбора данных. Все эти данные передаются в backend для последующего взаимодействия с ними в рамках клиентской части, реализующей ключевой функционал системы.

Физические устройства передают собранные данные в систему. Они включают как текстовую атрибутивную информацию, так и медиаконтент в виде фотоконтента или видеопотоков. Устройствами могут являться камеры фотофиксации, предоставляющие данные о зафиксированных событиях, а также видеокamеры, предоставляющие видеопоток.

Диаграмма разворачивания, визуализирующая взаимодействие компонентов системы, представлена на рисунке 2.

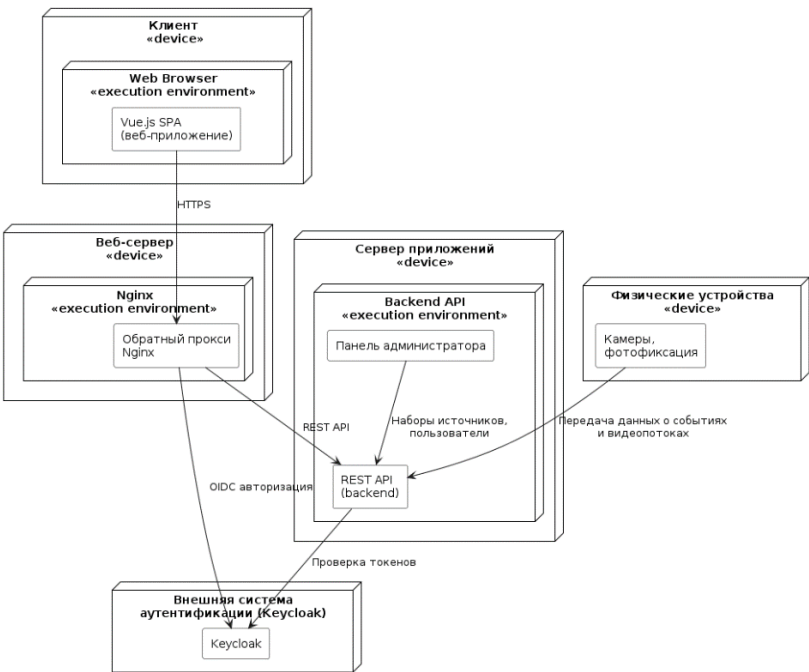


Рисунок 2 – Диаграмма разворачивания взаимодействия компонентов системы

В процессе проектирования архитектуры клиентской части приложения особое внимание было уделено модульности, инкапсуляции логики, удобству расширения и сопровождения реализованного функционала.

4.2 Структура компонентов

Визуальная и интерактивная часть клиентского приложения разработана посредством Vue-компонентов. Каждый из них отвечает за конкретную область пользовательского интерфейса и может содержать в себе более мелкие функциональные блоки. Они так же реализованы в виде компонентов, что позволяет поддерживать их атомарность и переиспользуемость. В конечном итоге, все блоки интерфейса собираются в крупные компоненты, представляющие собой страницы. В приложении их три:

- App – общий контейнер приложения;
- LoginPage – страница, реализующая функционал авторизации;
- MapSearchPage – страница, реализующая ключевой функционал системы.

Рассмотрим компоненты, непосредственно содержащие функциональные части пользовательского интерфейса:

- AuthForm – форма, содержащая логику локальной авторизации, а также авторизации через сервис внешней аутентификации Keycloak. Является частью LoginPage;
- MapContainer – интерактивная карта с отображаемыми наборами источников. Является частью MapSearchPage;
- SourcesModule – панель для работы с источниками на интерактивной карте, являющаяся частью страницы MapSearchPage. Она позволяет быстро взаимодействовать с источниками через панель, а также открывать панель событий и маршрутов для них;
- EventsModule – панель событий и маршрутов, предназначенная для отображения списка зафиксированных событий для событийных источников, а также списка маршрутов для подвижных объектов. Она напрямую связана с интерактивной картой;
- HeaderBlock – верхняя часть интерфейса, содержащий имя пользователя и действие выхода из аккаунта;
- LayerSwitcher – панель управления слоями интерактивной карты;

					ПГУ1.090501.08.001 ПЗ	Лист
						31
Из	Лист	№ докум.	Подпись	Дата		

- **DataPlayer** – подвижный растягиваемый медиаплеер для отображения фото- и видеоконтента, атрибутивной информации, предоставляемых источниками данных. Внутри себя отрисовывает интерфейс для работы с фотоконтентом посредством компонента **ImageViewTemplate**, и для видео – посредством **VideoViewTemplate**. Плеер может вызываться через соответствующие действия либо из компонента интерактивной карты, либо из панели источников, либо из панели событий и маршрутов.

4.3 Инкапсуляция бизнес-логики

Сложная клиентская бизнес-логика выносится в отдельные классы, доступ к которой осуществляется через их экземпляры. Это позволяет избежать перегрузки компонентов избыточным кодом, позволяя им отвечать только за представление данных, повторно использовать логику в разных частях интерфейса, а также упростить тестирование и отладку. Рассмотрим основные классы:

- **BaseLayerService** – отвечает за предоставление слоев картографических подложек;
- **StaticLayerService** – отвечает за представление слоев статичных объектов с применением клиентской кластеризации;
- **MovableLayerService** – отвечает за представление слоев подвижных объектов;
- **FeaturesPopupService** – отвечает за реализацию логики всплывающего окна с краткой информацией об объектах;
- **ContextMenuService** – отвечает за реализацию логики конструирования контекстного меню;
- **AreaSearchService** – отвечает за реализацию логики фильтрации источников по выделенной области. Связан с классом **ContextMenuService**, который предоставляет данный функционал.

4.4 Взаимодействие с сервером

Для обмена данными с бэкендом по REST API используется преднастроенный

					ПГУ1.090501.08.001 ПЗ	Лист
						32
Из	Лист	№ докум.	Подпись	Дата		

HTTP-клиент \$api. Он является экземпляром глобального и обеспечивает автоматическую подстановку базового URL API, подстановку токена доступа в заголовки HTTP-запросов, его обновление в случае истечения, а также перехват ошибок и глобальную их обработку.

Доступ к API происходит через специальные сервисные классы, реализующие набор статических методов, преимущественно в области одной сущности. Такой подход обеспечивает единый доступ к REST API, четкое разделение ответственности, улучшает читаемость и масштабируемость кода. Рассмотрим основные классы:

- AuthService – набор методов, содержащих логику отправки сетевых запросов, связанных с функционалом авторизации и аутентификации;
- KeycloakService – набор методов, содержащих логику отправки сетевых запросов, связанных с внешним сервисом аутентификации Keycloak;
- TileService – набор методов, содержащих логику получения картографических подложек;
- SourcesService – набор методов, содержащих логику получения наборов источников;
- VideoService – набор методов, содержащих логику получения видеопотока.

4.5 Глобальное хранилище данных

Для хранения глобальных данных, требующих использования в различных частях приложения, используется хранилище Vuex. В него сохраняется список наборов источников с заранее обработанными иконками, соответствующими необходимым параметрам.

4.6 Модульность

Клиентская часть приложения организована по модульному принципу, предполагающему логическое разделение проекта на функционально изолированные области. Такая архитектура повышает читаемость, упрощает

					ПГУ1.090501.08.001 ПЗ	Лист
						33
Из	Лист	№ докум.	Подпись	Дата		

сопровождение и масштабирование кода.

Рассмотрим основные каталоги:

- api – содержит сервисные классы для работы с сетевыми запросами, а также преднастроенный HTTP-клиент;
- assets – содержит статические ресурсы (картинки, шрифты, иконки);
- ui – содержит базовые переиспользуемые компоненты интерфейса;
- pages – содержит страницы приложения;
- store – глобальное хранилище приложения;
- plugins – инициализация и настройка сторонних библиотек;
- utils – вспомогательные утилитарные функции.

Такая структура кода соответствует принципам разделения ответственности и позволяет независимо развивать отдельные функциональные области без нарушения целостности системы.

Диаграмма компонентов, визуализирующая архитектуру клиентского приложения, представлена на рисунке 3.

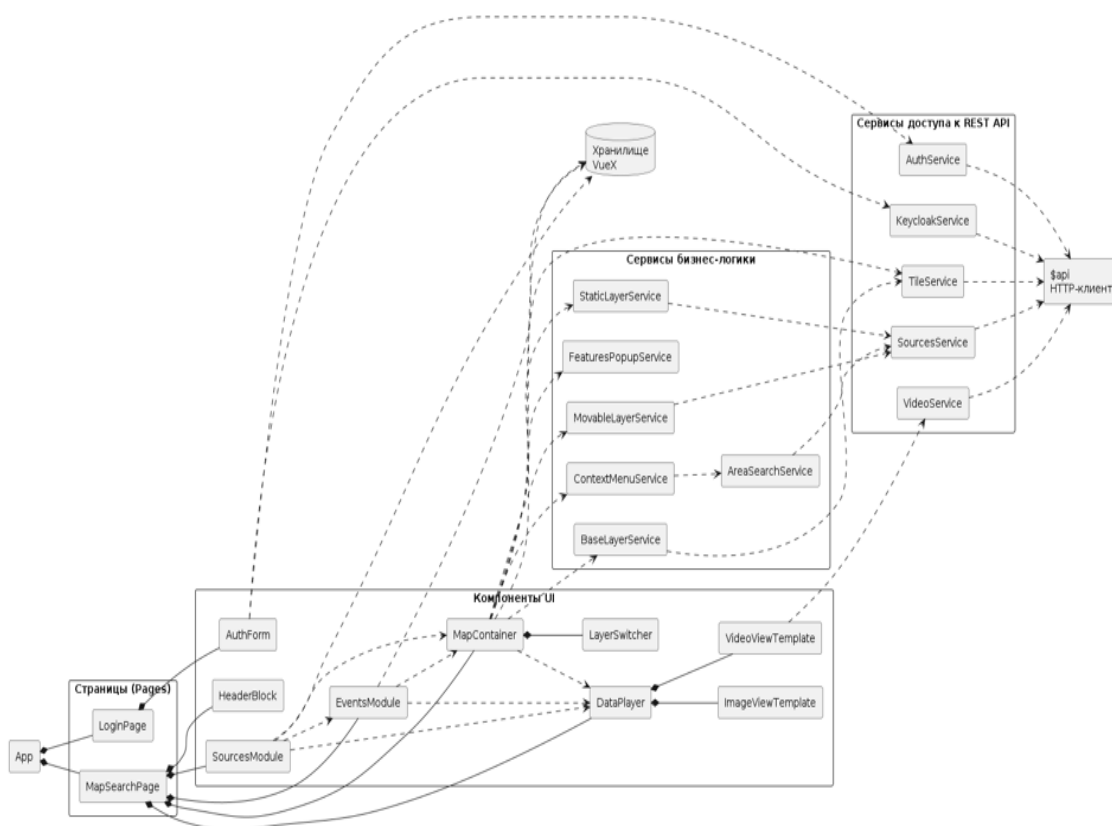


Рисунок 3 – Диаграмма компонентов архитектуры клиентского приложения

5 Реализация функциональных блоков

5.1 Авторизация, аутентификация и защита маршрутов

Целью реализации данного функционального блока является разработка защищенной пользовательской авторизации и аутентификации при работе с серверными ресурсами, альтернативной авторизации через внешний сервис аутентификации Keycloak, а также защиты маршрутов сайта от доступа к ним неавторизованных пользователей.

5.1.1 Технология аутентификации JSON Web Token

Для решения задачи защищенной пользовательской аутентификации при разработке системы используется архитектура, основанная на технологии JWT (JSON Web Token).

При прохождении авторизации пользователь отправляет свои учетные данные на сервер, и тот, в свою очередь, проверяя их валидность, возвращает две зашифрованных строки – токен доступа и токен обновления.

Токен доступа содержит зашифрованную информацию о пользователе и используется сервером для идентификации клиента в HTTP-запросах. Поскольку он имеет ограниченный срок подлинности, используется токен обновления, который отправляется клиентом для получения обновленного токена доступа. Данный механизм позволяет избежать непредвиденного завершения пользовательской сессии, заставляя пользователя проходить авторизацию повторно. Таким образом, происходит повышение безопасности клиент-серверного взаимодействия, дополнительно минимизируя риск компрометации токена за счет ограниченного срока жизни и при этом не нарушая активность пользовательской сессии.

Диаграмма последовательности механизма аутентификации JWT и взаимодействия HTTP-клиента с сервером представлена на рисунке 4.

					ПГУ1.090501.08.001 ПЗ	Лист
						35
Из	Лист	№ докум.	Подпись	Дата		

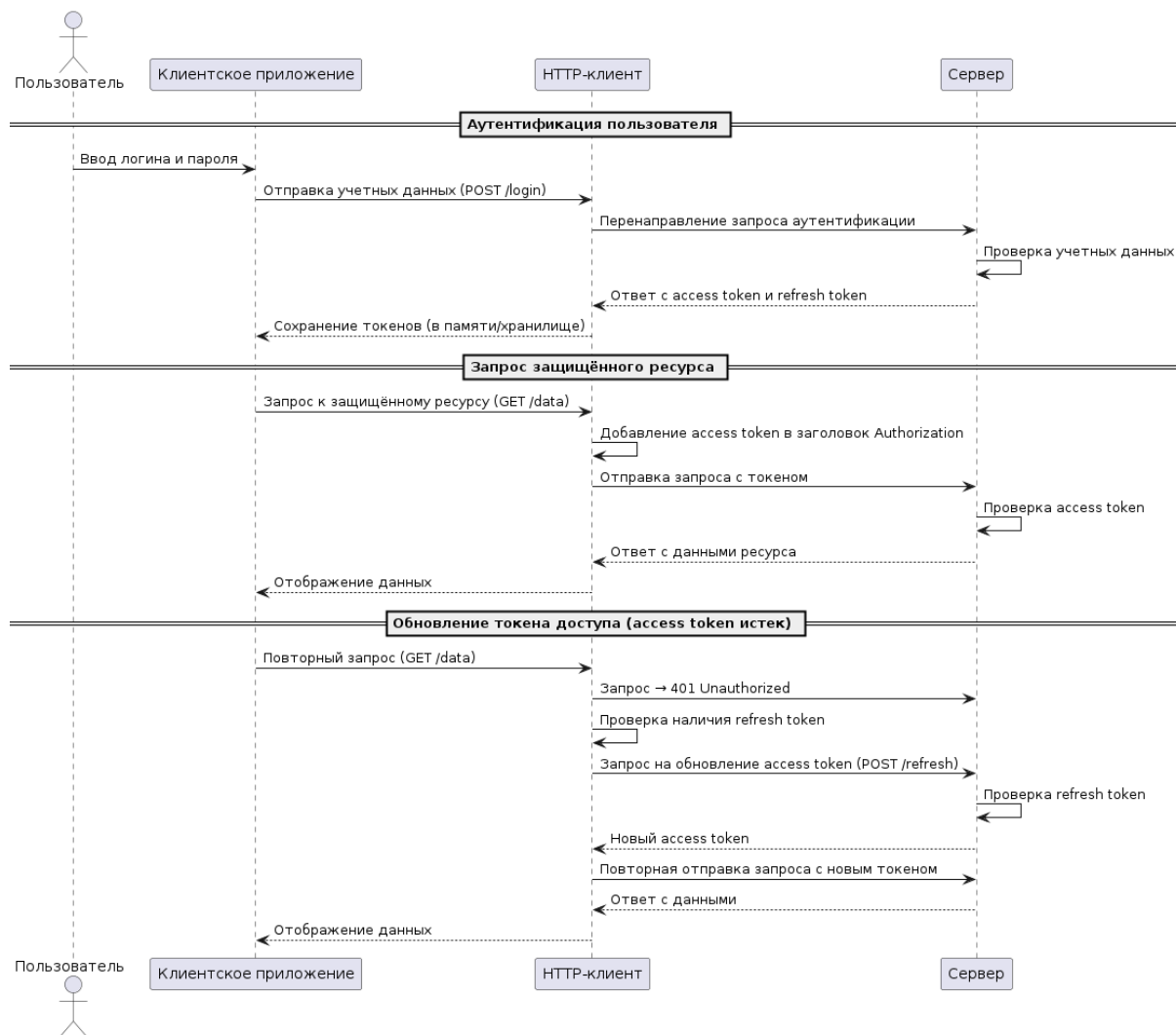


Рисунок 4 – Диаграмма последовательности механизма аутентификации JWT и взаимодействия HTTP-клиента с сервером

5.1.2 Сервисный класс сетевых запросов авторизации

Для обеспечения централизованной работы с сетевыми запросами, связанными с процессом аутентификации разработан специальный сервисный класс AuthService, где каждый из его статических методов реализует ту или иную операцию взаимодействия с сервером. Он инкапсулирует логику выполнения авторизации, обновления токена доступа, а также завершения пользовательской сессии. Рассмотрим основные методы.

Метод refreshToken выполняет операцию обновления токена доступа. Он выполняет PUT-запрос по маршруту /login/, передавая токен обновления в заголовке Authorization.

После получения обновленного токена происходит его замена в хранилище браузера `sessionStorage`, а также его возврат в исполняемый код.

```
static async refreshToken() {  
  const url = process.env.VUE_APP_API_URL + '/login/';
```

Метод `auth` выполняет авторизацию пользователя и принимает в качестве параметров учетные данные в виде логина и пароля. Операция реализуется по схеме HTTP Basic Auth, где логин и пароль кодируются в строку Base64 и после отправляется GET запрос по маршруту `/login/` с соответствующим заголовком. В случае успешной авторизации сервер возвращает токены доступа и обновления, которые помещаются в хранилище `sessionStorage`. Дополнительно в него устанавливается имя пользователя и тип авторизации.

```
static auth(login, password) {  
  const url = encodeURI(process.env.VUE_APP_API_URL +  
    '/login/');  
  const token =  
    Buffer.from(`${login}:${password}`).toString("base64");  
  return axios.get(url, {headers: {Authorization: "Basic "  
    + token}}).then(({data}) => {  
    const user = {  
      access_token: data.access_token,  
      refresh_token: data.refresh_token,  
      userName: login,  
    };  
    sessionStorage.setItem('user',  
      JSON.stringify(user));  
    sessionStorage.setItem('authService', 'local');  
  });  
}
```

Метод `logout` выполняет авторизацию завершения пользовательской сессии. Он выполняет отправку GET-запроса по маршруту `/logout/` с помощью преднастроенного HTTP-клиента `$api` (см. ниже). После завершения операции хранилище `sessionStorage`, а также глобальное хранилище `Vuex` очищаются.

```
static logout() {  
  return $api.get('/logout/').finally(() => {  
    sessionStorage.removeItem('user');  
    sessionStorage.removeItem('authService');  
  
    store.dispatch("sourceTypes/clearData");  
  });  
}
```

					ПГУ1.090501.08.001 ПЗ	Лист
						37
Из	Лист	№ докум.	Подпись	Дата		

```
}
```

5.1.3 Централизованный HTTP-клиент

Для взаимодействия клиента с сервером посредством библиотеки `axios` разработан централизованный HTTP-клиент. Его задачами является автоматическая подстановка токена доступа в заголовки сетевых запросов, автоматическое обновление в случае его истечения, а также глобальная обработка ошибок. Рассмотрим ключевые элементы реализации.

Для получения токена доступа из хранилища браузера разработана функция `getAccessToken`. Она используется для его извлечения из `sessionStorage`. Если токен доступа в нем отсутствует функция возвращает `null`.

```
function getAccessToken() {  
  const userStr = sessionStorage.getItem("user");  
  if (!userStr) return null;  
  try {  
    const user = JSON.parse(userStr);  
    return user?.access_token || null;  
  } catch (err) {  
    console.error("Error parsing user from sessionStorage:",  
err);  
    return null;  
  }  
}
```

Настраиваемый HTTP-клиент представляет собой экспортируемый объект `$api`, который является экземпляром глобального клиента `axios`. Он настроен с базовым URL API, который задан через переменные окружения, хранящиеся в объекте `process.env`. Данный HTTP-клиент используется во всем приложении вместо глобального вызова `axios`.

```
const API_URL = process.env.VUE_APP_API_URL;  
export const $api = axios.create({  
  baseURL: API_URL  
});
```

Перед каждым исходящим HTTP-запросом с клиента происходит автоматическое добавление токена доступа в заголовок `Authorization`. Это обеспечивает прозрачную идентификацию пользователя и информации о нем на сервере для отправки защищенных ресурсов. Логiku перехвата запросов реализует

					ПГУ1.090501.08.001 ПЗ	Лист
						38
Из	Лист	№ докум.	Подпись	Дата		

механизм interceptors.request.

```
$api.interceptors.request.use(  
  (config) => {  
    const token = getAccessToken();  
    if (token) {  
      config.headers.Authorization = `Bearer ${token}`;  
    }  
    return config;  
  },  
  (error) => {  
    return Promise.reject(error);  
  }  
);
```

Возврат сервером ответа на запрос клиента ошибки со статусом 401 Unauthorized означает то, что токен доступа истек и необходимо его обновление при помощи токена обновления. Логику перехвата ответов реализует механизм interceptors.response.

```
$api.interceptors.response.use(  
  (response) => {  
    return response;  
  },  
  async (error) => {  
    const originalRequest = error.config;  
  
    if (error.response?.status === 401 &&  
!originalRequest._isRetry) {  
      originalRequest._isRetry = true;  
      ...  
    }  
    return Promise.reject(error);  
  }
```

Для получения нового токена доступа происходит вызов метода refreshToken сервисного класса AuthService. После его получения все отложенные запросы на время обновления токена повторяют автоматически повторяют свой вызов с обновленным токеном. В случае, если обновление прошло с ошибкой, пользовательская сессия завершается и происходит перенаправление на страницу авторизации.

```
const newToken = await AuthService.refreshToken();  
isRefreshing = false;  
onRefreshed(newToken);  
originalRequest.headers.Authorization =
```

					ПГУ1.090501.08.001 ПЗ	Лист
						39
Из	Лист	№ докум.	Подпись	Дата		

```
`Bearer ${newToken}`;  
    return $api.request(originalRequest);
```

Если во время операции обновления токена доступа с клиента отправляются какие-либо запросы, они ожидают её завершения. Данная оптимизация позволяет избежать лишних обновлений. Каждый запрос подписывается через функцию `subscribeTokenRefresh` на завершение операции получения нового токена и повторяются, когда он будет получен.

```
let isRefreshing = false;  
let refreshSubscribers = [];  
function subscribeTokenRefresh(cb) {  
    refreshSubscribers.push(cb);  
}  
function onRefreshed(newToken) {  
    refreshSubscribers.forEach((cb) => cb(newToken));  
    refreshSubscribers = [];  
}
```

В случае, если токен обновления оказывается недействительным, выполняется очистка хранилища `sessionStorage`, в котором хранятся аутентификационные токены, а также происходит автоматический переход на страницу авторизации.

```
sessionStorage.clear();  
await router.push("/login");
```

Таким образом, данная реализация позволяет обеспечить централизованное управление состоянием пользовательской сессии, автоматическую авторизацию сетевых запросов, обновление токена без вмешательств пользователя, а также защиту от дублирования операций.

5.1.4 Локальная и внешняя авторизация Keycloak

Для обеспечения гибкости авторизации в системе используется два вида авторизации: локальная и через Keycloak. Локальная авторизация работает с пользователями базы данных собственной инфраструктуры, а авторизация через Keycloak работает с базой, расположенной на внешнем сервере.

Форма локальной авторизации разработана при помощи Vue.js и UI-фреймворка Vuetify. Каждое поле ввода при его изменении отдает событие `input`, на которое подписываются методы `setLogin` и `setPassword`, устанавливая результаты

					ПГУ1.090501.08.001 ПЗ	Лист
						40
Из	Лист	№ докум.	Подпись	Дата		

ввода учетных данных в объект formData для последующей работы с ним при нажатии кнопки входа в систему.

```

    <v-card-text>
      <HintedInput sign="Логин" color="primary-color"
class="mb-3" @input="setLogin" />
      <HintedInput sign="Пароль" color="primary-color"
@input="setPassword" :password="true" />
    </v-card-text>
    ...

    <v-card-actions style="display: flex; flex-direction: column;">
      <v-btn
        type="submit"
        color="primary-color"
        style="color: white; font-weight: 400;"
        block
      >
        Войти
      </v-btn>
      <template v-if="sharedData.keycloakEnabled">
        <div style="font-weight: 300;" class="my-2">ИЛИ</div>
        <v-btn
          color="#32718a"
          style="color: white; font-weight: 400;"
          block
          @click="authKeycloak"
        >
          Войти через Keycloak
        </v-btn>
      </template>
    </v-card-actions>

```

Форма локальной авторизации разработана при помощи Vue.js и UI-фреймворка Vuetify. Каждое поле ввода при его изменении отдает событие input, на которое подписываются методы setLogin и setPassword, устанавливая результаты ввода учетных данных в объект authData для последующей работы с ним при нажатии кнопки входа в систему.

После ввода своих учетных данных для локальной авторизации в системе необходимо нажать кнопку «Войти». Далее происходит вызов метода auth. Он осуществляет валидацию введенных значений в поля логина и пароля из authData и вызывает метод AuthService.auth. При успешной авторизации полученные с сервера токены доступа и обновления сохраняются и производится автоматический переход

					ПГУ1.090501.08.001 ПЗ	Лист
						41
Из	Лист	№ докум.	Подпись	Дата		

на основную страницу по маршруту /main.

```
auth() {  
  const {login, password} = this.authData;  
  if (this.authDataIsValid(login, password)) {  
    this.loading = true;  
    AuthService.auth(login, password).then(async () => {  
      this.$router.push('/main');  
    }).catch(e => {  
      errorHandlerNotify(e);  
    }).finally(() => this.loading = false);  
  } else {  
    this.$notify({  
      title: "Ошибка",  
      text: "Недопустимое имя пользователя или пароль",  
      type: "error"  
    });  
  }  
}
```

Страница локальной авторизации пользователя представлена на рисунке 5.

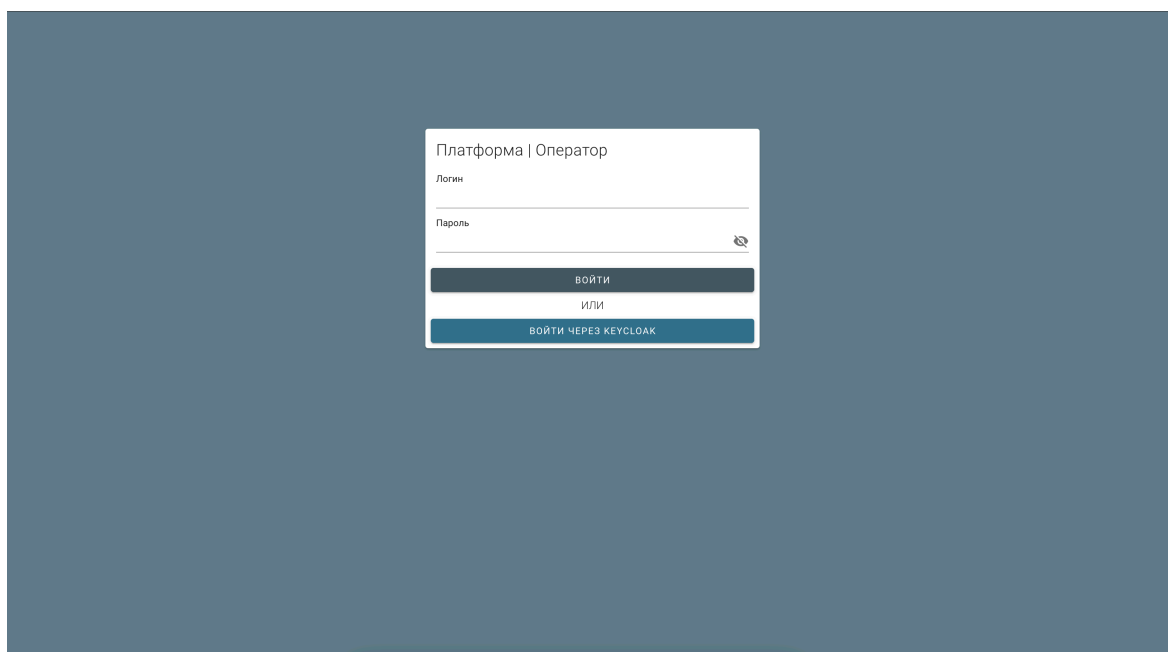


Рисунок 5 – Страница локальной авторизации пользователя

Если в системе включена поддержка Keycloak пользователю показывается кнопка авторизации через внешний провайдер. При нажатии данной кнопки происходит переход на внешнюю страницу Keycloak, где пользователь, введя свои учетные данные, автоматически возвращается обратно в приложение. В случае успешной авторизации сервер Keycloak отдает токен доступа, который

					ПГУ1.090501.08.001 ПЗ	Лист
						42
Из	Лист	№ докум.	Подпись	Дата		

обменивается на пару локальных токенов через соответствующий сетевой запрос к бэкенду. Далее доступ к сетевым ресурсам происходит стандартным образом через централизованный HTTP-клиент.

Диаграмма последовательности авторизации пользователя через Keycloak представлена на рисунке 6.

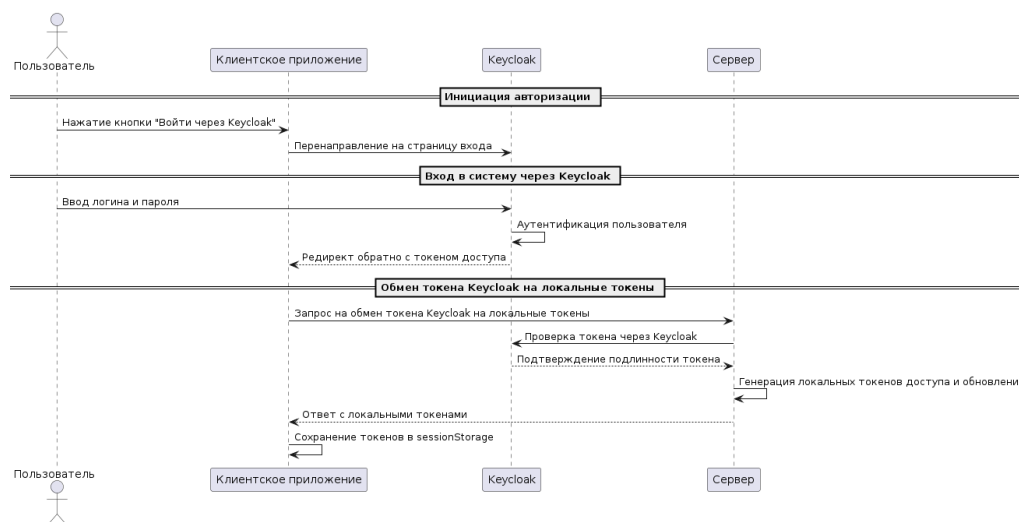


Рисунок 6 - Диаграмма последовательности авторизации пользователя через Keycloak

Для реализации данной логики был разработан специальный модуль для инициализации и обработки keycloak сессий. Он отвечает за конфигурацию, автоматический запуск аутентификации при заходе в приложение, обмен keycloak токена на пару локальных токенов, а также переход на основную страницу при успешной авторизации.

Функция `setKeycloakObject` отвечает за конфигурирование клиента на основе соответствующих данных, полученных с сервера. В ней происходит при помощи библиотеки `keycloak-js` происходит создание экземпляра `keycloak` клиента, который делается глобально доступным как `$keycloak` для дальнейшего взаимодействия с ним в различных частях приложения.

```

export const setKeycloakObject = (info) => {
  $keycloak = new Keycloak({
    url: info.url,
    realm: info.realm,
  })
}
  
```

```

        clientId: info.clientId
    });

```

Обработка события успешной аутентификации происходит в обработчике `onAuthSuccess` объекта `$keycloak`.

При успешной авторизации пользователь получает токен доступа `Keycloak`, который путем отправки GET-запроса по маршруту `/login/` обменивается на локальные токены. Полученные токены сохраняются в хранилище `sessionStorage`. Дополнительно устанавливается `authService` со значением `keycloak`. Данная логика необходима для определения типа авторизации пользователя. После успешной обработки происходит автоматический переход на основную страницу по маршруту `/main`.

```

$keycloak.onAuthSuccess = async () => {
    authProcess = true;
    sharedData.loadingApp = true;

    try {
        const url = process.env.VUE_APP_API_URL + '/login/';
        const {data} = await axios.get(url, {headers:
{Authorization: "Bearer " + $keycloak.token}});
        const user = {
            access_token: data.access_token,
            refresh_token: data.refresh_token,
            userName: 'keycloak_' +
$keycloak.tokenParsed.preferred_username,
        };
        await sessionStorage.setItem('user',
JSON.stringify(user));
        await sessionStorage.setItem('authService',
'keycloak');

        const currentRoute = router.currentRoute;
        if (currentRoute.path === '/login') await
router.push('/main');
        sharedData.loadingApp = false;
        authProcess = false;
    } catch (e) {
        if (e.response.status === 423) {
            sessionStorage.setItem('logoutMessage', 'no-
active');

            $keycloak.logout();
        }
    }
};

```

					ПГУ1.090501.08.001 ПЗ	Лист
						44
Из	Лист	№ докум.	Подпись	Дата		

Функция `initializeKeycloak` отвечает за инициализацию клиента Keycloak. При заходе в приложение осуществляется проверка включена ли авторизация через внешний провайдер путем вызова сервисного метода `KeycloakService.getWithoutAuth`. Если поле `keycloak_enabled`, пришедшее с сервера, имеет значение `true`, то происходит настройка клиента путем вызова функции `setKeycloakObject`, а также вызывается функция `init` у объекта `$keycloak` с опцией `check-sso` для проверки на то, имеется ли уже активная пользовательская сессия для автоматической авторизации пользователя.

Страница авторизации пользователя через Keycloak представлена на рисунке 7.

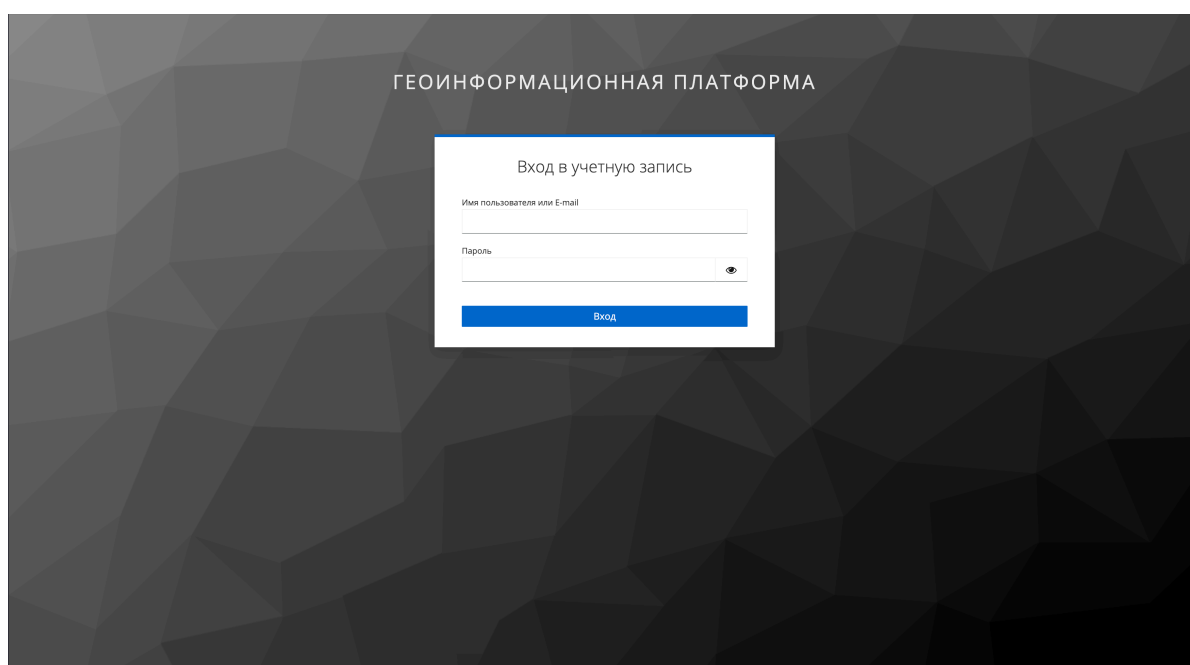


Рисунок 7 – Страница авторизации пользователя через Keycloak

5.1.5 Защита маршрутов

Для предотвращения доступа неавторизованных пользователей к закрытым маршрутам приложения разработан механизм фильтрации маршрутов на основе аутентификационных данных, хранящихся в данных сессии пользователя. Механизм реализован при помощи библиотеки `Vue Router` и глобального перехватчика переходов к маршрутам `beforeEach`.

Общая структура маршрутов в приложении определяется в массиве объектов

					ПГУ1.090501.08.001 ПЗ	Лист
						45
Из	Лист	№ докум.	Подпись	Дата		

routes. Каждому объекту присваивается флаг `meta.requiredAuth` который обозначает является ли данный маршрут закрытым. Основная логика валидации содержится в перехватчике `beforeEach`, который срабатывает перед каждым переходом на страницу.

Проверка активной сессии пользователя производится на основе наличия сессионных данных в хранилище `sessionStorage`.

Если маршрут защищен, то есть флаг `requiredAuth` равен `true`, то при наличии токена доступ к нему разрешается. Если авторизация осуществлялась через внешний провайдер, то дополнительно вызывается функция `initializeKeycloak` для восстановления сессии. В случае отсутствия токена пользователя перенаправляет на страницу авторизации `/login`.

```
router.beforeEach((to, from, next) => {
  const user = sessionStorage.getItem("user");
  const isAuth = !!user;
  if (to.path === '/login') initializeKeycloak();
  if (to.meta.requiresAuth) {
    if (isAuth) {
      if (sessionStorage.getItem("authService") ===
'keycloak') initializeKeycloak();
      next();
    } else next('/login');
  } else {
    if (to.path === '/login') {
      if (isAuth) next('/main')
      else next();
    } else {
      next();
    }
  }
})
```

5.2 Обработка источников данных и графических обозначений

За управление наборами источников отвечает отдельный модуль глобального хранилища данных `VueX`. Он отвечает за загрузку списка наборов источников с сервера, проверку наличия актуального локального кеша, обработку иконок при необходимости, а также сохранение результатов в хранилище и локальный кеш.

5.2.1 Получение информации о наборах источников с сервера

Получение списка наборов источников с информацией о них с сервера

					ПГУ1.090501.08.001 ПЗ	Лист
						46
Из	Лист	№ докум.	Подпись	Дата		

происходит путем вызова метода `getAll` разработанного сервисного класса `SourceTypeService`. Он инициирует GET-запрос по маршруту `/source_type/`, в результате которого приходит массив объектов, каждый из которых описывает соответствующий набор. Каждый объект включает следующие поля, характеризующие информацию о его иконке:

- `icon` – содержит URL-адрес изображения иконки;
- `icon_width`, `icon_height` – содержат ширину и высоту соответственно в пикселях, к которой необходимо преобразовать иконку на карте;
- `isCustomIcon` – характеризует необходимость обработки цвета иконки: если значение равно 0, то нет, а если 1, то да;
- `icon_color` – содержит цвет, к которому нужно привести иконку, в случае если `isCustomIcon` имеет значение 1.

5.2.2 Локальное кеширование

Для избежания лишних операций обновления иконки, в случае если параметры не изменились, разработан механизм локального кеширования информации о наборах источников. Он должен обрабатывать случаи перезагрузки страницы, либо перехода на другую. Хранение таких данных можно реализовать двумя способами: через технологию браузера `IndexedDB`, либо через хранилище `sessionStorage`.

`IndexedDB` – это низкоуровневый API браузера, которое позволяет хранить значительные объемы структурированных данных. В отличие от `sessionStorage`, `IndexedDB` поддерживает хранение сложных структур, таких как массивы объектов, без необходимости сериализации. Кроме того, оно не ограничено жестким лимитом в 5 – 10 мегабайт, как `sessionStorage`, что особенно важно при работе с объемными данными. Таким образом, для реализации локального кеширования данных лучшим образом подойдет именно данная технология.

Алгоритм работы локального кеширования содержит в себе несколько этапов.

Вначале, из каждого набора извлекаются параметры иконок, а именно `icon`, `icon_width`, `icon_height`, `icon_color` и `isCustomIcon`. Из данных значений составляется единая строка, представляющая собой хеш.

Далее, из базы данных `IndexedDB` происходит чтение предыдущих

					ПГУ1.090501.08.001 ПЗ	Лист
						47
Из	Лист	№ докум.	Подпись	Дата		

контрольных хеш-строк, а также списка обработанных иконок, представленных в виде Data URL.

Если сохраненная хеш-строка совпадает с текущей и в кеше есть соответствующие обработанные иконки, то они загружаются из базы и применяются к текущим наборам. В случае, когда хеш совпадает, но иконки отсутствуют, происходит их обработка и затем происходит сохранение. Если хеш не совпадает, то это означает что параметры изменились и требуется повторная обработка иконок с последующей заменой их в кеше.

Таким образом, реализуется баланс между производительностью и точностью работы алгоритма, сводя к минимуму избыточные вычисления.

Диаграмма деятельности кеширования иконок представлена на рисунке 8.

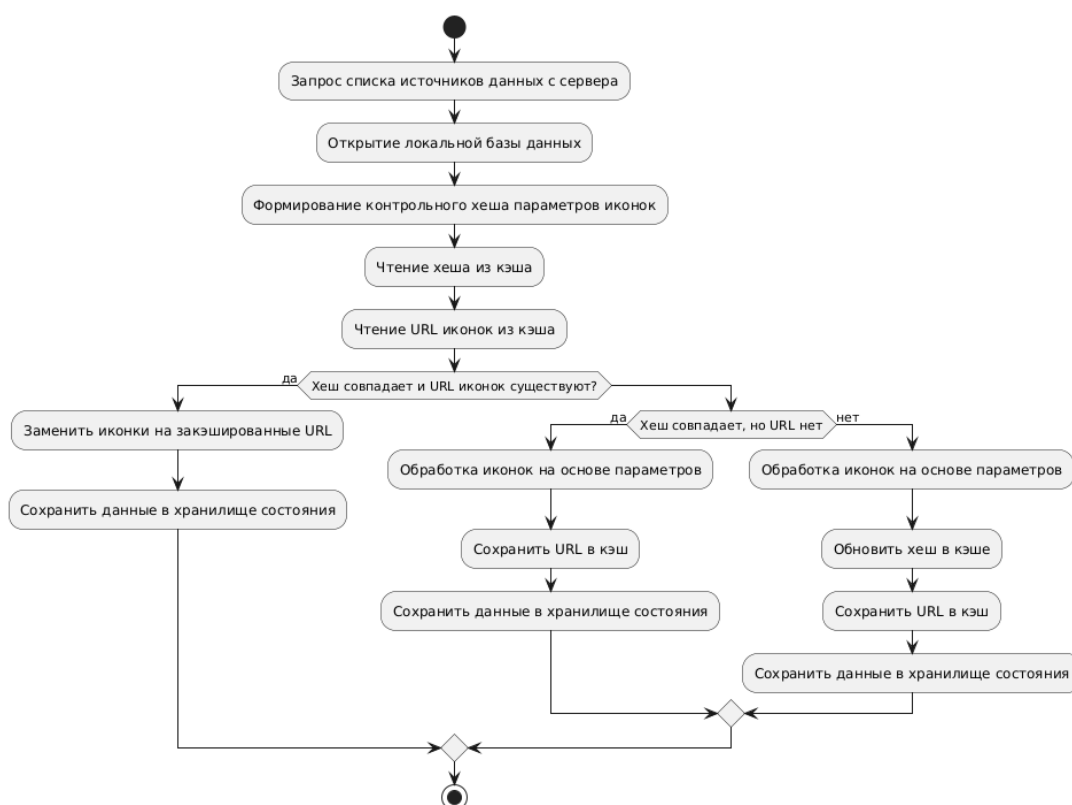


Рисунок 8 – Диаграмма деятельности кеширования иконок

Для реализации данного алгоритма были разработаны несколько вспомогательных функций для работы с хранилищем. Они предназначены для инкапсулирования логики открытия базы данных, чтения и обновления данных.

Функция `openDatabase` инициирует открытие базы `iconsDatabase`. При первом открытии автоматическим образом инициализируются её структура, а именно два хранилища объектов: `hashes` для хранения контрольной строки с параметрами иконок и `icons` для хранения URL-адресов иконок. Данная логика выполняется в обработчике события `onupgradeneeded`. В случае успешного выполнения функции возвращается объект базы данных для последующего взаимодействия с ним.

```
function openDatabase() {
  return new Promise((resolve, reject) => {
    const request = indexedDB.open('iconsDatabase', 1);
    request.onupgradeneeded = (e) => {
      const db = e.target.result;
      if (!db.objectStoreNames.contains('hashes')) {
        db.createObjectStore('hashes', {keyPath:
'key'}));
      }
      if (!db.objectStoreNames.contains('icons')) {
        db.createObjectStore('icons', {keyPath: 'key'});
      }
    };
    request.onsuccess = (e) => resolve(e.target.result);
    request.onerror = (e) => reject(e.target.error);
  });
}
```

Функция `getFromDB` реализует функционал чтения из хранилища по ключу. Она принимает три аргумента, а именно объект базы данных, имя хранилища объектов и ключ, по которому происходит поиск. В функции создается транзакция на чтение и осуществляется выборка значения по ключу. В случае успеха возвращается либо содержимое поля, либо `null`, если запись отсутствует. Данный функционал обеспечивает доступ к ранее сохраненным хешам и спискам URL для принятия решения о повторной обработке данных.

```
function getFromDB(db, storeName, key) {
  return new Promise((resolve, reject) => {
    const transaction = db.transaction([storeName],
'readonly');
    const store = transaction.objectStore(storeName);
    const request = store.get(key);
    request.onsuccess = () => resolve(request.result ?
request.result.value : null);
    request.onerror = (e) => reject(e.target.error);
  });
}
```

					ПГУ1.090501.08.001 ПЗ	Лист
						49
Из	Лист	№ докум.	Подпись	Дата		

```
}
```

Функция updateDB реализует функционал записи или обновления данных в хранилище. Она принимает четыре аргумента, а именно объект базы данных, имя хранилища, ключ и значение для записи. В функции создается транзакция readwrite, в которой вызывается метод put, обеспечивающий как новую запись, так и обновление существующей.

```
function updateDB(db, storeName, key, value) {  
  return new Promise((resolve, reject) => {  
    const transaction = db.transaction([storeName],  
'readwrite');  
    const store = transaction.objectStore(storeName);  
    const request = store.put({key, value});  
    request.onsuccess = () => resolve();  
    request.onerror = (e) => reject(e.target.error);  
  });  
}
```

После получения списка наборов источников с сервера система выполняет серию действий по минимизации вычислений для подготовки их к визуализации в пользовательском интерфейсе.

Для каждого набора источников извлекаются поля, характеризующие внешний вид иконки, а именно icon, icon_width, icon_height, icon_color, isCustomIcon. Для этих значений формируется контрольная строка, представляющая собой их конкатенацию. В последующем, она используется для сравнения с ранее сохраненным значением в локальном хранилище.

Из базы данных IndexedDB извлекаются ранее сохраненная хеш-строка, а также массив URL-адресов иконок. После выполнения их сравнения по алгоритму, описанному выше, объектам наборов источников присваиваются актуальные URL иконок и данные сохраняются в глобальное хранилище VueX.

```
const {data} = await SourceTypesService.getAll();  
const db = await openDatabase();  
const hashes = data  
  .map(  
    (item) =>  
    `${item.icon}_${item.icon_height}_${item.icon_width}_${item.isCu  
stomIcon}_${item.icon_color}`  
  )
```

					ПГУ1.090501.08.001 ПЗ	Лист
						50
Из	Лист	№ докум.	Подпись	Дата		

```

        .join(',');
        const cachedParamsHashes = await getFromDB(db,
'hashes', 'icon_hashes');
        const iconsUrls = await getFromDB(db, 'icons',
'icons_url');
        if (cachedParamsHashes && cachedParamsHashes ===
hashes) {
            if (iconsUrls) {
                data.forEach((el, i) => (el.icon =
iconsUrls[i]));
                commit('setData', data);
            } else {
                const processedData = await
processIcons(data);
                await updateDB(db, 'icons', 'icons_url',
processedData.map((el) => el.icon));
                commit('setData', processedData);
            }
        } else {
            const processedData = await processIcons(data);
            await updateDB(db, 'hashes', 'icon_hashes',
hashes);
            await updateDB(db, 'icons', 'icons_url',
processedData.map((el) => el.icon));
            commit('setData', processedData);
        }
    }
}

```

5.2.3 Обработка иконок

Перед отображением иконок наборов источников они проходят обработку в соответствии с заданными для них параметрами: размера и цвета. Данная логика реализована внутри функции `processIcons`. Она принимает массив объектов с информацией о наборах источников и преобразует URL-иконок в Data URL, хранящие непосредственно сами обработанные изображения для их немедленного использования в различных частях приложения. Функция работает с двумя форматами иконок: PNG и SVG. Для PNG устанавливается размер, для SVG – размер и цвет. Для избежания блокировки пользовательского интерфейса и эффективного использования сетевых ресурсов обработка изображений происходит асинхронно и параллельно внутри метода `Promise.all` из API JavaScript.

Рассмотрим основные утилитарные функции:

- `isSvg(url)` – принимает URL изображения и определяет расширение путём его анализа;

					ПГУ1.090501.08.001 ПЗ	Лист
						51
Из	Лист	№ докум.	Подпись	Дата		

- `fetchText(url)` и `fetchBlob(url)` – асинхронно загружают содержимое изображения как текст для формата SVG и как бинарный объект для формата PNG соответственно;
- `svgDataURL(txt)` – преобразует SVG-строку в формат `data:image/svg+xml` с кодировкой UTF-8 для использования обработанного изображения в процессе разработки пользовательского интерфейса;
- `makeCanvas(w, h)` – создаёт графическую поверхность Canvas либо OffscreenCanvas в случае его поддержки используемым браузером для более оптимизированного рендеринга;
- `recolorRaster(ctx, w, h, color)` – применяет цветовой фильтр к изображению, расположенному на графической поверхности, используя режим смешивания `source-in`. Таким образом обеспечивается наложение цвета поверх иконки при сохранении её формы.

Во время работы основной функции происходит обработка каждого элемента массива, возвращая новый объект с обновленным полем `icon`, содержащим закодированное изображение. Работа алгоритма разбита на две части: для формата SVG и для формата PNG.

При работе с иконкой формата SVG она загружается в виде текста. Далее, из корневого тега `<svg>` происходит удаление исходных параметров `width`, `height` и `preserveAspectRatio` для возможности в дальнейшем задать им обновленные значения вручную. На основе параметров ширины и высоты, заданных в информации о наборе источников, в текст внедряются соответствующие атрибуты ширины и высоты. Если поле `isCustomIcon` набора сигнализирует о том, что иконке необходимо задать цвет, в код SVG-изображения внедряется дополнительный блок `<style>`, который переопределяет все цвета элементов через принудительное изменение свойств `fill` и `stroke`, отвечающих за заливку и обводку. После выполнения данных операций иконка кодируется в Data URL.

При работе с иконкой формата PNG она загружается в виде бинарного изображения и создается `ImageBitmap` - специально оптимизированное браузером представление изображения, предназначенное для быстрого помещения на

					ПГУ1.090501.08.001 ПЗ	Лист
						52
Из	Лист	№ докум.	Подпись	Дата		

графическую поверхность Canvas или OffscreenCanvas. После, происходит создание поверхности и отрисовка исходного изображения на ней.

Если поле isCustomIcon набора сигнализирует о том, что иконке необходимо задать цвет, графическая поверхность перекрашивается через функцию recolorRaster. Далее происходит сериализация полученного изображения, а именно, если доступен метод OffscreenCanvas.convertToBlob – создается специальный blob URL, иначе используется стандартный canvas.toDataURL.

После выполнения данных операций функция processIcons возвращает обновленный массив объектов с уже обработанными иконками готов для немедленного использования в пользовательском интерфейсе.

Диаграмма деятельности функции обработки иконок processIcons представлена на рисунке 9.

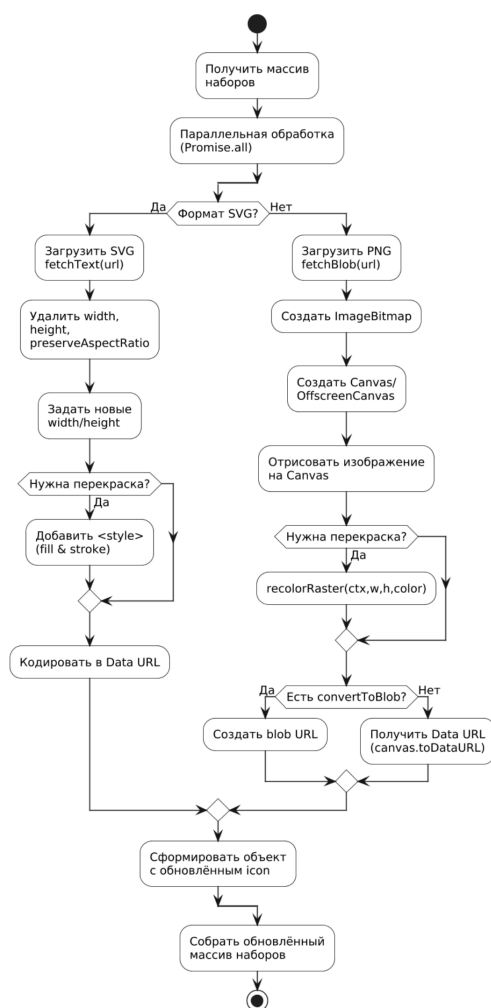


Рисунок 9 - Диаграмма деятельности функции обработки иконок processIcons

5.3 Инициализация карты и картографических подложек

5.3.1 Конструирование слоев подложек

Получение списка картографических подложек с информацией о них осуществляется через метод `getBaseLayers` сервисного класса `TileService`. Он отправляет GET-запрос по маршруту `/geotile/saved/`.

Для работы с картографическими подложками по протоколу TMS разработан сервисный класс `BaseLayerService`. Он предназначен для конструирования слоев карты и их управления. Конструктор класса принимает название картографической подложки, а также её URL-адрес.

Для конструирования слоя используется метод `createTMSLayer`. Он предназначен для создания и возврата объекта, представляющего собой тайловый слой с тайловым источником XYZ. В методе конструируется объект, содержащий название подложки и источник тайлов. Источник представляет собой экземпляр класса XYZ из библиотеки `OpenLayers`, в конструктор которого передается URL-адрес. Картографическая подложка состоит из множества маленьких квадратных картинок, представляющих собой тайлы. Они динамически загружаются по мере масштабирования и прокрутки карты. Используемый класс XYZ определяет шаблон, по которому будет формироваться URL-адрес для каждого тайла на основе базового адреса, управляет кэшированием, а также автоматически подставляет в адрес каждого тайла необходимые значения координат и масштабов для их автоматической подгрузки. Далее создается слой, представляющий собой экземпляр класса `TileLayer`, в конструктор которого передается сконструированный объект с настройками.

```
createTMSLayer(data) {  
    const {title, url} = data;  
    if (!title || !url) return;  
    const options = {  
        title: title,  
        source: new XYZ({  
            url: url + '/{z}/{x}/{-y}.png',  
        }),  
    };  
}
```

					ПГУ1.090501.08.001 ПЗ	Лист
						54
Из	Лист	№ докум.	Подпись	Дата		

```

        return new TileLayer(options);
    }

```

Полученный слой возвращается и становится публично доступным через геттер `getLayer`. Для получения его видимости, а также её изменения, разработаны методы `getVisibility` и `setVisibility`, которые управляют слоем через API библиотеки `OpenLayers` `getVisible` и `setVisible` соответственно.

```

getLayer() {
    return this.layer;
}
setVisibility(visibility) {
    this.layer.setVisible(visibility);
}
getVisibility() {
    return this.layer.getVisible();
}

```

5.3.2 Инициализация карты

Инициализация карты осуществляется путем создания экземпляра `OpenLayers` класса `Map`. В качестве контейнера для данного блока интерфейса используется DOM-элемент, доступ к которому осуществляется через механизм `Vue ref`. Основные параметры, указываемые в конструкторе класса, включают:

- элементы управления – добавляется управление полноэкранным режимом, масштаб карты через кнопки и линию прокрутки через экземпляры соответствующих классов `FullScreen`, `ZoomSlider` и `ScaleLine`. Для отображения в правом нижнем углу карты координат, соответствующих курсору мыши, используется экземпляр класса `MousePositionControl`, в конструктор которого передаётся проекция, а именно `EPSG:4326`, а также CSS-класс, содержащий набор стилей для отображения;
- пользовательские взаимодействия – в дополнение к стандартным средствам взаимодействия с картой, добавляется собственный обработчик изменения масштаба карты, при помощи которого масштабирование карты происходит более плавно и ограничивает переходы между уровнями масштаба;
- параметры отображения – задаются характеристики в виде начального положения центра карты, уровня масштабирования, а также проекция карты в виде

					ПГУ1.090501.08.001 ПЗ	Лист
						55
Из	Лист	№ докум.	Подпись	Дата		

EPSG:3857, так как она является наиболее распространённой для поставщиков картографических тайлов, а также задаются границы данной проекции для предотвращения случаев заикливания карты по горизонтали.

Добавление слоев на карту осуществляется через вызов метода `addLayer` полученного экземпляра класса с аргументом в виде объекта слоя.

```
this.map = new Map({
  controls: defaultControls().extend([
    new FullScreen(),
    new ZoomSlider(),
    new ScaleLine(),
    this.mousePositionControl(),
  ]),
  target: this.$refs.mapContainer,
  interactions: defaults().extend([mouseWheelZoom]),
  layers: [],
  view: new View({
    center: fromLonLat([37.684324, 55.818154]),
    zoom: 14,
    constrainOnlyCenter: false,
    extent: [-20026376.39, -20048966.10, 20026376.39,
20048966.10],
    projection: new Projection({
      code: 'EPSG:3857',
      units: 'm',
      global: false,
      extent: get('EPSG:3857').getExtent()
    })
  })
});
```

При изменении размера контейнера, в который помещена карта, необходимо автоматическим образом отслеживать это событие и перерисовывать карту. Для реализации данной логики используется объект `ResizeObserver`.

Он отслеживает изменение размера DOM-элемента и вызывает функцию обновления размеров самой карты при помощи метода `updateSize`.

5.4 Отображение статичных источников

Для реализации алгоритма отрисовки статичных источников на карте была разработана собственная технология клиентского отображения, обеспечивающая оптимизированную и производительную работу браузера в условиях большого

					ПГУ1.090501.08.001 ПЗ	Лист
						56
Из	Лист	№ докум.	Подпись	Дата		

количества объектов или их высокой плотности расположения. Рассмотрим основные методы работы.

Для минимизации сетевой нагрузки, а также нагрузки на браузер пользователя с сервера для каждого набора запрашиваются только те источники, которые попадают в видимую область. Полученные данные собираются в единой структуре данных, которая проходит процедуру клиентской кластеризации. Она объединяет объекты, находящиеся в пределах заданного радиуса, в один кластерный объект, который при приближении масштаба раскрывается на более мелкие. Данная операция позволяет избежать случая наложения объектов в зонах их высокой плотности, повысить производительность за счет минимизации отрисовываемых источников, а также сделать восприятие картографических данных пользователем более простым. После выполнения операции кластеризации полученные обработанные данные группируются в один векторный слой. Это дает возможность централизованно управлять их отображением, стилизацией, а также унифицированно работать с пользовательскими событиями. Для обработки случаев, когда источники находятся слишком близко друг к другу, применяется механизм принудительной раскластеризации если масштаб карты достаточно маленький. Для источников, которые не попали в кластер или для которых произошла принудительная раскластеризация возможно три типа отображения:

- одиночный источник – отображается в виде иконки, соответствующей его набору;
- несколько источников одного и того же набора в одной точке – отображается в виде иконки, соответствующей его набору, с числовым идентификатором в её левом нижнем углу;
- несколько источников, принадлежащих разным наборам в одной точке – отображается в виде центральной точки, от которой равноудалённо отходят линии. На их концах находятся иконки соответствующих наборов. Если источников для набора несколько, то дополнительно отображается числовой идентификатор.

Диаграмма деятельности отображения статичных источников представлена на рисунке 10.

					ПГУ1.090501.08.001 ПЗ	Лист
						57
Из	Лист	№ докум.	Подпись	Дата		

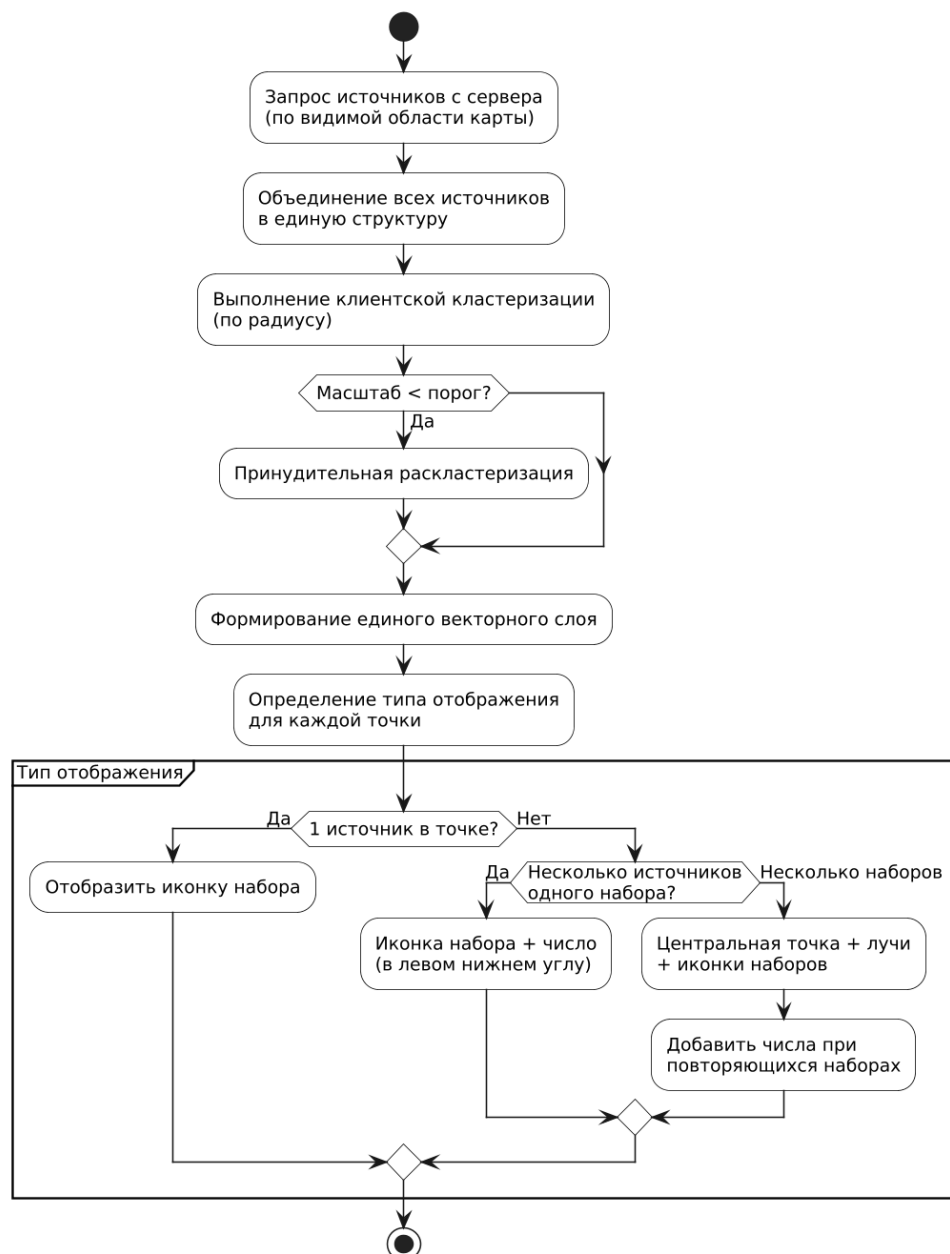


Рисунок 10 - Диаграмма деятельности отображения статичных источников

Для оптимизации сетевого взаимодействия наборы источников запрашиваются с сервера только после прохождения 300 мс после конца движения карты. Это значение оптимальной незаметной задержки для пользователя.

В случае, если во время отправки HTTP-запросов на получение источников для наборов пользователь снова произвел движение карты, необходимо отменить отправленные HTTP-запросы для последующей их переотправки для обновленной видимой области карты. Диаграмма последовательности сетевого взаимодействия запроса статичных источников с сервера представлена на рисунке 11.

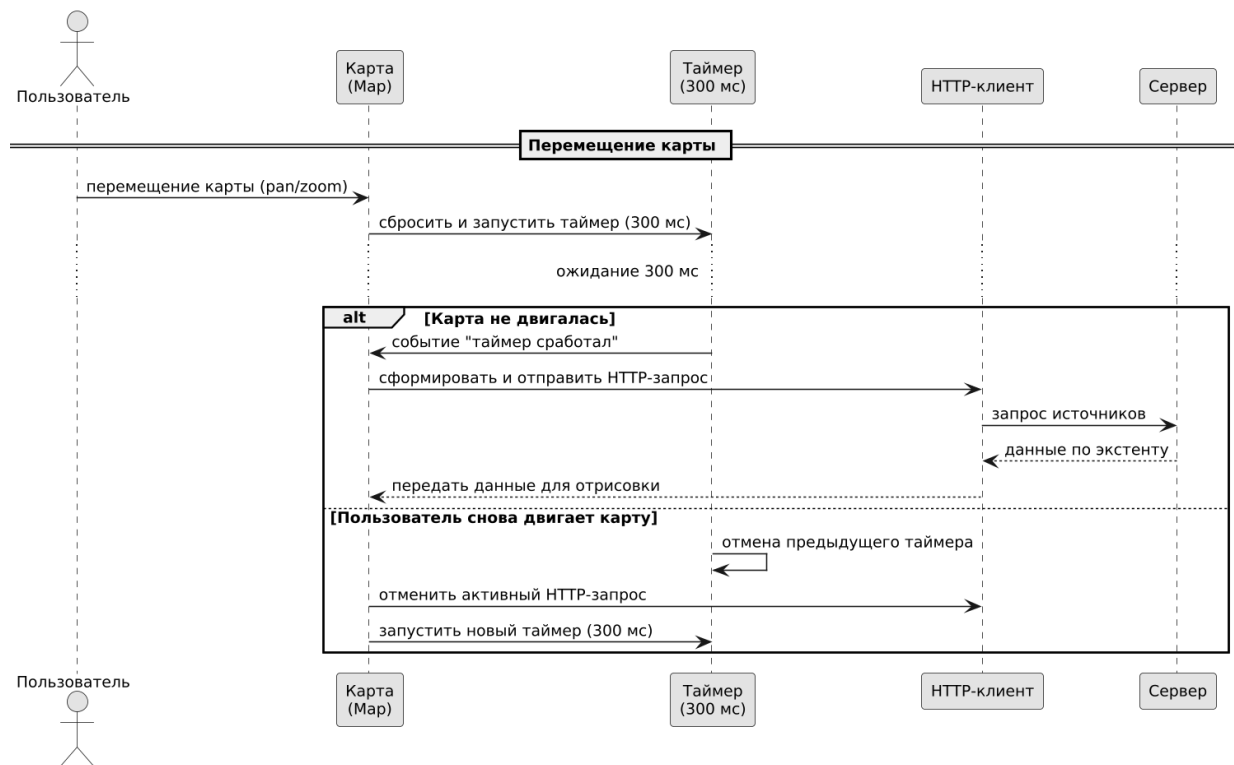


Рисунок 11 - Диаграмма последовательности сетевого взаимодействия запроса статичных источников с сервера

5.4.1 Слой статичных объектов

Для реализации логики отображения статичных источников был разработан сервисный класс StaticLayerService. Его конструктор принимает массив объектов, хранящий информацию о наборах источников статичного типа. Экземпляр данного класса агрегирует все переданные данные об объектах в один векторный слой. Для осуществления доступа к нему разработан геттер getLayer.

Для получения текущей видимой области карты – экстенета, используется утилитарная функция getExtentForMapView. Она принимает в качестве аргумента ссылку на объект карты. В функции происходит расчет прямоугольника в проекции карты EPSG:3857, соответствующего видимой области, и, путем преобразования, возврат значения в проекции координат EPSG:4326. Данная проекция подходит для использования её в процессе клиент-серверного взаимодействия.

```

export const getExtentForMapView = (map) => {
  let extent = map.getView().calculateExtent(map.getSize());
  return transformExtent(extent, 'EPSG:3857', 'EPSG:4326');
}
  
```


Для получения объектов источников данных с сервера используется метод `getObjects` сервисного класса `SourcesService`. Он позволяет получить список источников для необходимого набора с учетом фильтрации их только по переданной прямоугольной области. Дополнительно, метод принимает объект `AbortSignal`, позволяющий при необходимости отменить данный HTTP-запрос после его отправки.

```
static getObjects(sourceTypeId, ext, maxPointsCount, signal) {
    const URLparams =
`xmin=${ext[0]}&ymin=${ext[1]}&xmax=${ext[2]}&ymax=${ext[3]}`;
    const url =
`/geocluster/${sourceTypeId}/${maxPointsCount}/?${URLparams}`;
    return $api.get(url, {signal});
}
```

Для отслеживания передвижения картографической области пользователем для объекта карты устанавливается слушатель события `moveend`, который срабатывает каждый раз после того, как движение карты закончено и прошло 300 мс. В его обработчике при помощи утилитарной функции `getExtentForMapView` рассчитываются координаты видимой области и вызывается метод обновления объектов на карте `updateFeaturesOnMap`.

Метод `updateFeaturesOnMap` выполняет перерисовку видимой области карты. Вначале происходит отмена запущенных сетевых операций на получение источников, используя метод `abort` объекта `AbortController` из API JavaScript. Далее, из массива наборов источников извлекаются только те наборы, которые помечены как видимые и извлекаются их идентификаторы. Из видимых наборов источников формируется массив асинхронных операций, сформированный при помощи метода `getObjects` сервисного класса `SourcesService`, после чего в методе `Promise.allSettled` происходит их выполнение. Если хотя бы один из запросов завершился отменой, функция прерывается и возвращает пустое значение – `undefined`. После получения результатов асинхронных операций со списками источников, они группируются в единый массив и обрабатываются функцией кластеризации `clusterFeatures`. После завершения всех операций полученный список объектов отрисовывается на векторном слое, путем помещения их в объект типа `VectorSource`.

					ПГУ1.090501.08.001 ПЗ	Лист
						60
Из	Лист	№ докум.	Подпись	Дата		

5.4.2 Клиентская кластеризация

Для внедрения механизма клиентской кластеризации источников была разработана функция `clusterFeatures`. Библиотека `OpenLayers` предлагает готовый инструмент в виде объекта `Cluster`, но его использование оказалось нецелесообразным в рамках специфики разрабатываемого приложения.

На основе реализуемой технологии отображения большого количества объектов на карте данные загружаются только по видимой области, а не для всей поверхности. При передвижении области карты или изменении масштаба на сервер отправляется соответствующий запрос, занимающий определенное время.

При использовании готового механизма клиентской кластеризации, предлагаемого библиотекой `OpenLayers`, для видимых устаревших объектов, которые еще не успели обновиться, происходит автоматической перерасчет кластеров. После того, как обновленные данные с сервера пришли, они повторно перерисовываются на карте.

Пользователь видит двойную смену кластеров, а именно сначала для устаревших данных, затем для обновленных. Это вызывает эффект «скачков» или «дергания» объектов, что отрицательно влияет на восприятие картографической информации.

Для устранения данной проблемы было принято решение разработать собственную функцию кластеризации, которая будет обрабатывать источники единственный раз сразу же после их прихода с сервера перед отрисовкой. В таком случае, после движения карты объекты останутся в таком же виде до прихода обновленных данных.

Функция `clusterFeatures` принимает в качестве параметров список объектов типа `Features`, а также ссылку на экземпляр карты. Она осуществляет кластеризацию на основе пиксельного расстояния. Алгоритм работы заключается в следующем.

Для каждого объекта вычисляются его координаты и преобразуются в пиксельные при помощи метода `getPixelForCoordinate`. В результате формируется

					ПГУ1.090501.08.001 ПЗ	Лист
						61
Из	Лист	№ докум.	Подпись	Дата		

соответствующий массив pixelCoords. Он содержит пиксельные координаты для каждого объекта в формате (x, y) пригодные для сравнения;

```
const pixelCoords = [];
for (let i = 0; i < features.length; i++) {
  const geometry = features[i].getGeometry();
  if (!geometry) continue;
  const coords = geometry.getCoordinates();
  if (Array.isArray(coords)) {
    pixelCoords.push(map.getPixelFromCoordinate(coords));
  } else {
    pixelCoords.push(null);
  }
}
```

Сравнение расстояний происходит на основе расчета евклидова расстояния d по формуле

$$d^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2, \quad (1)$$

где x_1, x_2 – координаты первого объекта;

y_1, y_2 – координаты второго объекта.

По сравнению с классическим видом используется квадрат расстояния d , что позволяет избежать лишней операции вычисления корня увеличивая производительность.

```
function getPixelDistanceSquared(pixel1, pixel2) {
  if (!pixel1 || !pixel2) return Infinity;
  const dx = pixel1[0] - pixel2[0];
  const dy = pixel1[1] - pixel2[1];
  return dx * dx + dy * dy;
}
```

Основной цикл алгоритма итерируется по каждому объекту и группирует объекты в кластеры, которые находятся в пределах заданного радиуса, а именно 35 пикселей. Те объекты, которые уже сгруппированы, исключаются из обработки через множество processed.

```
for (let i = 0; i < features.length; i++) {
  if (processed.has(features[i])) continue;

  const cluster = [];
```

					ПГУ1.090501.08.001 ПЗ	Лист
						62
Из	Лист	№ докум.	Подпись	Дата		

```

        const centerPixel = pixelCoords[i];
        if (!centerPixel) continue;

        for (let j = 0; j < features.length; j++) {
            if (processed.has(features[j])) continue;

            const targetPixel = pixelCoords[j];
            if (getPixelDistanceSquared(centerPixel, targetPixel)
<= clusterRadiusSquared) {
                cluster.push(features[j]);
                processed.add(features[j]);
            }
        }
    }
}

```

Основной цикл алгоритма итерируется по каждому объекту и группирует объекты в кластеры, которые находятся в пределах заданного радиуса, а именно 35 пикселей. Те объекты, которые уже сгруппированы, исключаются из обработки через множество `processed`.

```

for (let i = 0; i < features.length; i++) {
    if (processed.has(features[i])) continue;
    const cluster = [];
    const centerPixel = pixelCoords[i];
    if (!centerPixel) continue;
    for (let j = 0; j < features.length; j++) {
        if (processed.has(features[j])) continue;
        const targetPixel = pixelCoords[j];
        if (getPixelDistanceSquared(centerPixel, targetPixel)
<= clusterRadiusSquared) {
            cluster.push(features[j]);
            processed.add(features[j]);
        }
    }
}

```

Если в кластере более одного объекта, создается новый объект `Feature`, центр которого вычисляется как среднее арифметическое всех координат внутри него. Если объект не попал ни в один кластер он возвращается в исходном виде.

```

function getClusterCenter(coordinates) {
    const sum = coordinates.reduce((acc, coord) => {
        acc[0] += coord[0];
        acc[1] += coord[1];
        return acc;
    }, [0, 0]);
    return [sum[0] / coordinates.length, sum[1] /
coordinates.length];
}
const clusterFeature = new Feature({

```

					ПГУ1.090501.08.001 ПЗ	Лист
						63
Из	Лист	№ докум.	Подпись	Дата		

```

geometry: new Point(clusterCenter),
features: cluster,
size: cluster.length,
});

```

Модуль интерфейса, отображающий наборы статичных источников на картографической подложке представлен на рисунке 12.

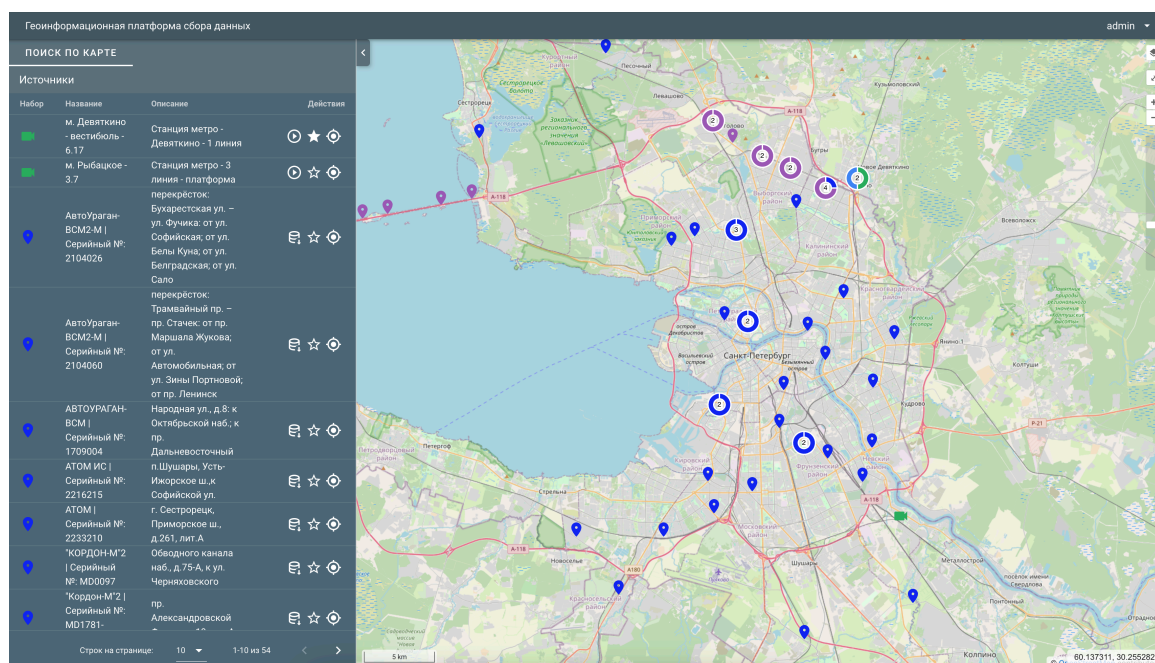


Рисунок 12 - Модуль интерфейса, отображающий наборы статичных источников на картографической подложке

5.4.3 Стилизация объектов на карте

Логика стилизации объектов инкапсулирована в классе `StaticLayerUtils`. Для векторного слоя статичных источников разработан централизованный обработчик стилей `styleHandler`.

Он автоматическим образом вызывается при отрисовке каждого объекта на карте и на основе информации о нем возвращает тот или иной объект стилей `Style`. Рассмотрим основные типы визуализации.

Кластерные объекты. Если объект является кластером, содержащим вложенные источники, логика отрисовки зависит от текущего уровня масштаба карты. Обработка масштабов карты происходит по следующему принципу:

- если уровень масштаба карты больше 15, выполняется принудительная

раскластеризация всех вложенных объектов. Так как масштаб карты маленький, в большинстве случаев пользователь сможет визуально различить объекты даже при их высокой плотности расположения. Для обработки случаев, когда несколько источников находятся в одной точке, перед отрисовкой на карте происходит их обработка при помощи функции `categorizeFeaturesByPosition`. Она производит их группировку по признаку нахождения в одной и той же координате. Для объектов, расположенных в разных точках, применяется метод `commonObject`. Он предоставляет объект стилей для одиночного источника, включая иконку соответствующего набора и установку якорной точки привязки координат. Для объектов, расположенных в одной и той же точке, применяется метод `branchesObject`. Он производит их визуализацию в виде равноудалённых лучей-линий, исходящих из центра — координат этой группы;

- если уровень масштаба карты меньше или равен 15, то вызывается метод `clusterObject`, предоставляющий объект стилей для стандартного кластера. Он представляет собой окружность, сегменты которой покрашены в цвета соответствующих наборов, а в центре находится числовой идентификатор в виде количества вложенных источников внутри.

Несколько источников в одной точке, принадлежащих одного набору. Если объект содержит несколько источников в одной точке, принадлежащих одному набору, вызывается метод `moreSourcesObject`. Он предоставляет объект стилей, представляющий собой иконку набора с отображающимся числовым идентификатором количества источников.

Одиночный источник. Если источник не был сгруппирован в кластер и его координаты не совпадают с координатами других объектов, то вызывается метод `commonObject`, возвращающий стандартный объект стилей.

Метод `clusterObject` отвечает за стилизацию кластерных объектов. Они принимаются соответствующим параметром `features` в виде массива. В методе для него вызывается вспомогательная функция `calcFeaturesData`, подсчитывающая количество объектов каждого набора и соответствующих им цветов из поля `icon_color`.

					ПГУ1.090501.08.001 ПЗ	Лист
						65
Из	Лист	№ докум.	Подпись	Дата		

Рассчитанные значения передаются в объект круговой диаграммы из библиотеки ol-ext. Она отрисовывается через объект Chart и настраивается через параметры радиуса radius, обводки stroke и заливки fill.

Центр кластера содержит числовую подпись общего количества объектов через использование объекта Text. Для избежания выхода текста за границы объекта происходит его динамическое масштабирование в зависимости от длины.

```
clusterObject(features) {
    const featuresDataByST = this.calcFeaturesData(features);
    const allSum = featuresDataByST.data.reduce((sum, value)
=> sum + value, 0);

    return new Style({
        image: new Chart({
            type: "donut",
            data: featuresDataByST.data,
            radius: 17,
            colors: featuresDataByST.colors,
            stroke: new Stroke({
                color: "#fff",
                width: 2
            }),
            fill: new Fill({
                color: "white"
            })
        }),
        text: new Text({
            text: String(allSum),
            fill: new Fill({
                color: "black"
            }),
            backgroundFill: new Fill({
                color: "white"
            }),
            scale: 1.1 - (String(allSum).length / 10)
        })
    })
}
```

Метод commonObject отвечает за стилизацию одиночных источников и дополнительно может использоваться в более сложных структурах стилей. Для набора, которому принадлежит объект, извлекается иконка, ширина, высота и якорная точка привязки в пикселях.

На основе ширины, высоты и якорной точки рассчитывается нормализованное значение якоря в виде доли от ширины и высоты. Установка иконки с якорем

					ПГУ1.090501.08.001 ПЗ	Лист
						66
Из	Лист	№ докум.	Подпись	Дата		

происходит через соответствующий объект Icon.

```
commonObject(feature, member) {
  const sourceType = feature.get('sourceType');
  const sourceTypeInfo = this.sourceTypesInfoList.find(
    info => info.sourceType === sourceType
  );
  if (!sourceTypeInfo?.icon) {
    console.warn('Иконка не найдена для типа источника:',
sourceTypeInfo);
    return new Style({});
  }
  const { icon, icon_width, icon_height, icon_x, icon_y } =
sourceTypeInfo;
  const iconOptions = { src: icon };
  if (!member) {
    iconOptions.anchor = [icon_x / icon_width, icon_y /
icon_height];
  }
  const styleOptions = {
    image: new Icon(iconOptions)
  };
  if (member) {
    styleOptions.geometry = feature.getGeometry();
  }
  return new Style(styleOptions);
}
```

Вид отображения кластерных объектов представлен на рисунке 13.

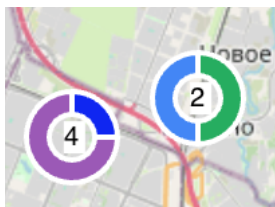


Рисунок 13 - Вид отображения кластерных объектов

Метод `moreSourcesObject` отвечает за стилизацию источников одного набора, располагающихся в одной точке. Отличием от предыдущего метода является дополнительное добавление числового идентификатора к иконке через объект `Text`.

```
const textStyle = new Text({
  text: count,
  font: 'bold 12px sans-serif',
  fill: new Fill({ color: '#000' }),
  stroke: new Stroke({ color: '#fff', width: 2 }),
  offsetX: -icon_width / 2 + 5,
```

					ПГУ1.090501.08.001 ПЗ	Лист
						67
Из	Лист	№ докум.	Подпись	Дата		

```

        offsetY: icon_height / 2 - 5,
        textAlign: 'left',

        textBaseline: 'bottom'
    });
    const styleOptions = {
        image: iconStyle,
        text: textStyle
    };

```

Вид отображения источников одного набора, располагающихся в одной точке, представлен на рисунке 14.

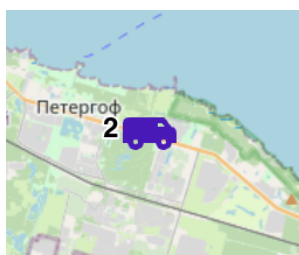


Рисунок 14 - Вид отображения источников одного набора, располагающихся в одной точке

Метод `moreSourcesObject` отвечает за стилизацию группы источников, располагающихся в одной точке и принадлежащим различным наборам источников. Для исключения ситуации наложения нескольких иконок друг на друга был разработан механизм, позволяющий оптимизированно визуализировать данный случай при помощи равноудаленных линий от центра группы. Сначала вызывается функция `generatePointCircle`, которая рассчитывает расположение точек в виде координат по окружности вокруг центра. Для избежания слияния линий при расчете учитывается количество объектов и масштаб карты. Каждый объект отрисовывается на рассчитанных координатах и соединяется с центром тонкой линией `LineString`.

В зависимости от состава объекта к нему применяется соответствующий метод стилизации `commonObject` или `moreSourcesObject`. В место соединения линий для удобства восприятия центра пользователем помещается небольшая точка радиусом 3 пикселя.

					ПГУ1.090501.08.001 ПЗ	Лист
						68
Из	Лист	№ докум.	Подпись	Дата		

Утилитарная функция `generatePointCircle` предназначена для вычисления равномерно расположенных координат точек по окружности, исходя из заданного центра, количества объектов и масштаба карты.

Алгоритм работы включает в себя определение равных угловых шагов на основе количества точек по окружности; вычисления радиуса окружности, обеспечивающего визуальное расстояние между объектами в 35 пикселей; на основе центра и угла рассчитывает координаты для каждой точки на основе тригонометрических функций; возвращает массив новых координат.

```
branchesObject(features, resolution) {
  return generatePointsCircle(
    features.length,
    features[0].getGeometry().getCoordinates(),
    resolution,
  ).reduce((styles, coordinates, i) => {
    const feature = features[i];
    feature.set('branchPositionOnMap', coordinates);
    const isMoreSourcesObject =
feature.get('ids')?.length > 1 && feature.get('object_count') > 1;
    const isCommonSource = feature.get('ids') &&
feature.get('object_count') === 1;
    const point = new Point(coordinates);
    const line = new
LineString([features[0].getGeometry().getCoordinates(),
coordinates]);
    styles.unshift(
      new Style({
        geometry: line,
        stroke: new Stroke({
          color: 'rgb(0,0,0)',
          width: 1,
        }),
      }),
    );
  });
  export const generatePointsCircle = (count, clusterCenter,
resolution) => {
    const circleDistanceMultiplier = 1;
    const circleFootSeparation = 28;
    const circleStartAngle = Math.PI / 2;
    const circumference =
      circleDistanceMultiplier * circleFootSeparation * (2 +
count);
    let legLength = circumference / (Math.PI * 2);
    const angleStep = (Math.PI * 2) / count;
    const res = [];
    let angle;
    legLength = Math.max(legLength, 35) * resolution;
```

					ПГУ1.090501.08.001 ПЗ	Лист
						69
Из	Лист	№ докум.	Подпись	Дата		

```

    for (let i = 0; i < count; ++i) {
        angle = circleStartAngle + i * angleStep;
        res.push([
            clusterCenter[0] + legLength * Math.cos(angle),
            clusterCenter[1] + legLength * Math.sin(angle),
        ]);
    }
    return res;
}

```

Вид отображения группы источников, располагающихся в одной точке и принадлежащим различным наборам источников, представлен на рисунке 15.

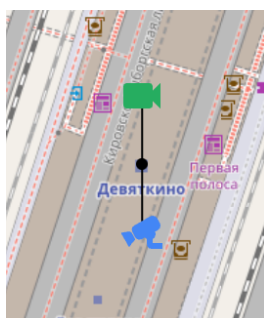


Рисунок 15 - Вид отображения группы источников, располагающихся в одной точке и принадлежащим различным наборам источников

5.5 Отображение подвижных источников

В рамках данного функционала отображения подвижных наборов источников предусмотрена возможность отображения архивных маршрутов передвижения объекта на карте и реализация отслеживания местоположения источника с обновлением в реальном времени. Данная логика инкапсулирована внутри сервисного класса `MovableLayerService`, который принимает информацию о наборе, конструирует слой и возвращает его через соответствующий геттер `getLayer`.

По умолчанию на карте отображается последнее местоположение для каждого источника в наборе. За реализацию данной логики отвечает метод `renderLayer`, в котором происходит запрос информации о последнем маршруте с сервера. Далее, из полученных данных извлекается финальная точка траектории для каждого объекта, которая отображается на карте в виде иконки набора.

Помимо этого, к ней прикрепляются атрибутивные данные об источнике для

					ПГУ1.090501.08.001 ПЗ	Лист
						70
Из	Лист	№ докум.	Подпись	Дата		

быстрого просмотра информации, например, при наведении на неё курсора мыши во всплывающем окне. В случае, если для текущего набора задано обновление данных в реальном времени, вызов метода `renderLayer` помещается в функцию `setInterval` из API JavaScript с заданным количеством секунд, который производит его автоматический вызов и перерисовку слоя.

За визуализацию выбранного маршрута на картографическом слое отвечает метод `showTrackOnMap`. Он принимает его уникальный идентификатор, на основе которого с сервера получает информацию о нём. Для формирования траектории движения на карте используется объект `LineString`, представляющий собой линию, в который передаются все её точки. Для её стилизации используется метод `line` сервисного класса `MovableLayerStyles`, возвращающий определенный набор стилей.

Логика работы метода `line` заключается в возврате массива стилей, представляющих собой линию, покрашенную в цвет, заданный для соответствующего набора.

Для повышения читаемости на фоне картографической подложки осуществляется дополнительная обводка черным цветом по краям. Дополнительно, по середине каждого сегмента траектории располагается иконка стрелочки, которая показывает направление движения. Для этого извлекается соответствующий массив координат и на основе него вычисляется средняя точка и угол наклона для располагаемой иконки.

```
new Style({
  geometry: new Point(midpoint),
  image: new Icon({
    src: LSArrow,
    anchor: [0.75, 0.5],
    color: '#000000',
    crossOrigin: 'anonymous',
    rotateWithView: true,
    rotation: -rotation,
  }),
})
calcRotationMidpoint(start, end) {
  const dx = end[0] - start[0];
  const dy = end[1] - start[1];
```

					ПГУ1.090501.08.001 ПЗ	Лист
						71
Из	Лист	№ докум.	Подпись	Дата		

```

const midpoint = [start[0] + dx / 2, start[1] + dy / 2];
return {rotation: Math.atan2(dy, dx), midpoint};
}

```

Вид отображения подвижных наборов источников представлен на рисунке 16.



Рисунок 16 - Вид отображения подвижных наборов источников

5.6 Интерфейс отображения событий, маршрутов и источников данных

Панель источников располагается слева от интерактивной карты и по умолчанию отображает видимые пользователем источники на карте.

Помимо этого, она используется для отображения результатов фильтрации объектов по выделенной картографической области, а также списка сгруппированных источников при взаимодействии пользователя с ними.

При движении карты пользователем происходит автоматическая отправка глобального события `onChangeExtent` с координатами текущей видимой прямоугольной области. Его обработчик располагается непосредственно в самом Vue компоненте панели.

Он принимает данные о координатах области, конструирует запрос с методом GET на основе них и отправляет его. Для оптимизации сетевого взаимодействия перед отправкой нового запроса происходит отмена старых, в случае если они

					ПГУ1.090501.08.001 ПЗ	Лист
Из	Лист	№ докум.	Подпись	Дата		72

выполняются. Для каждого источника доступны следующие действия: центрирование карты по выбранному источнику, открытие панели просмотра событий для событийных источников, открытие панели просмотра маршрутов для подвижных источников, открытие видеоплеера для потоковых источников. Панель отображения источников представлена на рисунке 17.

ПОИСК ПО КАРТЕ			
Источники			
Набор	Название	Описание	Действия
	м. Девяткино - вестибюль - 6.17	Станция метро - Девяткино - 1 линия	
	Радио	Test	
	АвтоУраган-ВСМ2-М Серийный №: 2104026	перекрёсток: Бухарестская ул. – ул. Фучика; от ул. Софийская; от ул. Белы Куна; от ул. Белградская; от ул. Сало	
	АвтоУраган-ВСМ2-М Серийный №: 2104060	перекрёсток: Трамвайный пр. – пр. Стачек; от пр. Маршала Жукова; от ул. Автомобильная; от ул. Зины Портновой; от пр. Ленинск	
	АВТОУРАГАН-ВСМ Серийный №: 1709004	Народная ул., д.8: к Октябрьской наб.; к пр. Дальневосточный	
	АТОМ ИС Серийный №: 2216215	п. Шушары, Усть-Ижорское ш., к Софийской ул.	
	АТОМ Серийный №: 2233210	г. Сестрорецк, Приморское ш., д.261, лит.А	
	"КОРДОН-М"2 Серийный №: MD0097	Обводного канала наб., д.75-А, к ул. Черняховского	
	"Кордон-М"2 Серийный №: MD1701	пр. Александровской	
Строк на странице: 10 1-10 из 63 < >			

Рисунок 17 – Панель отображения источников

Панель просмотра событий и маршрутов предназначена для отображения связанных с источником событий или маршрутов. Данный элемент интерфейса динамически открывается в нижней части карты с возможностью его закрытия пользователем. Для каждого события включает функционал просмотра связанного

фотоконтента, а для маршрутов – просмотр видеозаписи или визуализация на карте. Панель просмотра событий и маршрутов представлена на рисунке 18.

Данные								
КоАП	Дата возникновения события	ГРЗ	Скорость ТС, км/ч	Название источника	Описание источника	Широта	Долгота	Тип транспорта
Нет данных	19.12.2022 01:11:13	H508PA750	75	АвтоУраган-ВCM2-M Серийный №: 2104026	перекрёсток: Бухарестская ул. – ул. Фучика; от ул. Софийская; от ул. Белы Куна; от ул. Белградская; от ул. Сало	59.87882	30.37317	легковой автомобиль

Строк на странице: 10 1-1 из 1

Рисунок 18 – Панель просмотра событий и маршрутов

5.7 Фильтрация по выделенной области

Для упрощения взаимодействия с большими наборами данных на карте предусмотрен функционал фильтрации источников по произвольно выделенной области пользователем. Эта функция позволяет отобразить в левой панели видимых источников только те, которые попали внутрь нарисованного полигона.

Данное действие доступно через контекстное меню интерактивной карты. При нажатии правой кнопкой мыши открывается список действий, в котором необходимо выбрать пункт «Создать выделение». При наведении на него открывается вложенное меню с доступными формами, среди которых имеется действие рисования полигона.

После выбора пользователю предоставляется возможность рисования на карте замкнутой области по ключевым точкам. После рисования система автоматически определяет границы полигона и отправляет соответствующий запрос к серверу.

После его завершения в левой панели происходит обновление списка источников на отображение только объектов, попавших в нарисованную область.

Дополнительно, действия отправки запроса на сервер и отображения результатов выводятся пользователю в виде всплывающих уведомлений с соответствующими сообщениями. Также, имеется возможность передвигать нарисованную область мышью с повторной переотправкой запроса к серверу. Если

пользователю больше не требуется данный функционал он может выйти из режима открыв контекстное меню и выбрав пункт «Убрать выделение».

Логика работы контекстного меню инкапсулирована внутри сервисного класса ContextMenuService. Дополнительно к функционалу работы с выделением области конструируются быстрые действия для работы с картой, а именно центрирование, уменьшение и увеличение масштаба по месту курсора. Пункты меню «Создать выделение» и «Убрать выделение» дополнительно связаны с сервисным классом AreaSearchService, который содержит в себе логику работы с областями и их обработки.

При выборе пункта создания выделения происходит добавление векторного слоя VectorLayer. Далее, создается объект Draw, с помощью которого пользователь может создавать на карте различные геометрические фигуры. В обработчике события завершения рисования drawend вызывается метод viewSourcesForArea и при помощи вызова метода enableTranslateForCurrentFeature активируется возможность перемещения области мышью.

```
this.drawInteraction.on('drawend', (event) => {  
    this.disableDrawing();  
    this.currentFeature = event.feature;  
    this.currentFeature.set('type', type);  
    this.viewSourcesForArea();  
    this.enableTranslateForCurrentFeature();  
});
```

Метод viewSourcesForArea отвечает за выполнение фильтрации источников по выделенной области. Он вызывается сразу после завершения рисования фигуры и обеспечивает запрос и отображение результата в виде списка источников.

В методе из нарисованной фигуры полигона извлекаются координаты соединительных точек полигона и переводятся из формата проекции карты EPSG:3857 в географическую проекцию EPSG:4326. На основе полученных данных осуществляется формирование SQL-запроса, который будет отправлен в качестве параметра на сервер. Для формирования условия, позволяющего отобрать все источники, чья геометрия полностью входит в указанный полигон используется пространственная функция ST_Within, в которую передаются координаты.

					ПГУ1.090501.08.001 ПЗ	Лист
						75
Из	Лист	№ докум.	Подпись	Дата		

```
const whereCondition = `ST_Within(geometry,
ST_GeomFromText('POLYGON(${coordinatesInString}))', 4326))`;
```

После выполнения запроса список отфильтрованных источников выводится в левой панели путем вызова глобального события onCustomViewSources.

Функционал фильтрации источников по выделенной области представлен на рисунке 19.

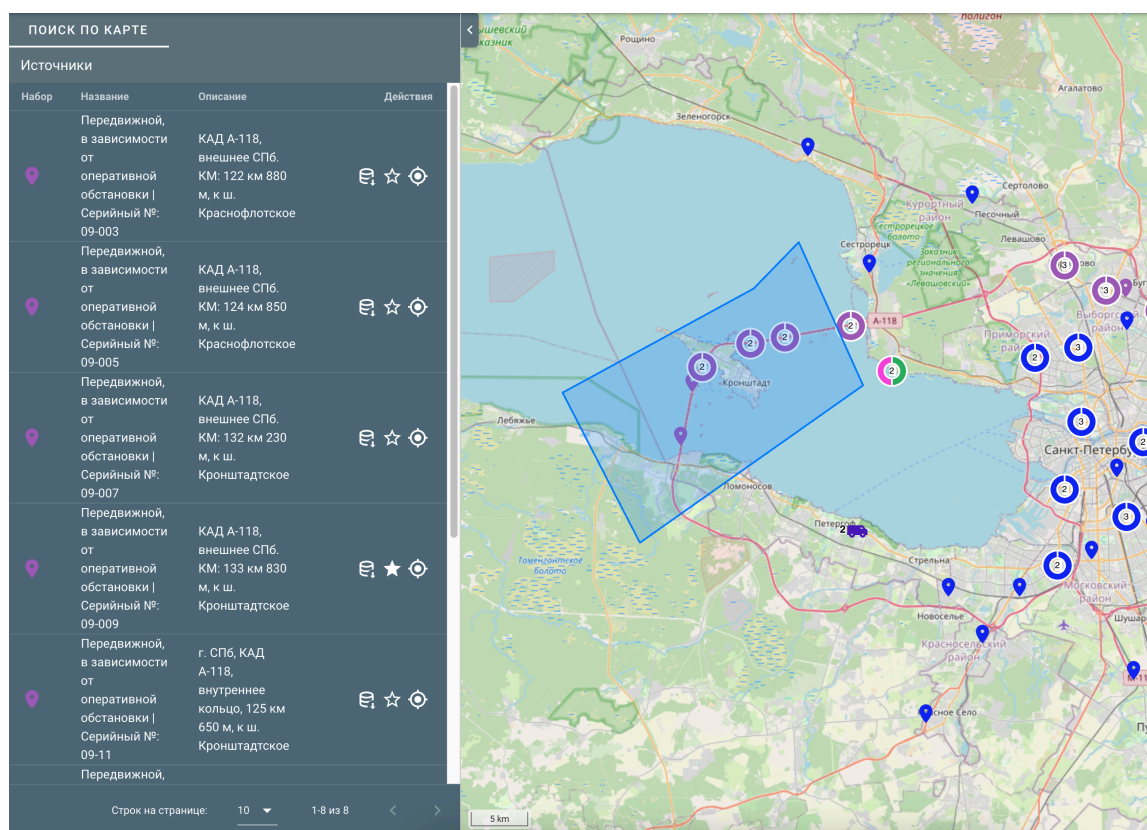


Рисунок 19 - Функционал фильтрации источников по выделенной области

5.8 Медиаплееры

В рамках разработки приложения имеется встроенная возможность работы с медиаплеерами для просмотра фото- или видеоконтента, связанных с определенным источником данных.

Для управления данным функционалом разработан специальный компонент-обертка, который располагается поверх всего пользовательского интерфейса. Он отвечает за динамическое создание экземпляров компонента медиаплеера, добавление их в DOM-дерево и автоматическое удаление при закрытии.

За создание отвечает метод `createComponent`. В нём при помощи метода `extend` из API Vue создается экземпляр базового компонента `DataPlayer`, представляющего собой перетаскиваемое окно с возможностью изменения его размеров. Далее, при помощи метода `$mount` происходит его монтирование в DOM-дерево и регистрируется обработчик события `close`, по которому происходит закрытие плеера.

```
const ComponentClass = Vue.extend(DataPlayer);
const instance = new ComponentClass({vuetify, store,
propsData: {data}});
instance.$mount();
this.$refs.container.appendChild(instance.$el);
this.instances.push(instance);
instance.$on('close', () => {
  this.removeInstance(instance);
});
```

5.8.1 Фотоплеер

Фотоплеер предназначен для отображения прикрепленных к событию фотографий, а также содержит панель действий, через которую можно сохранить контент или дополнительно просмотреть атрибутивные данные.

Центральную область занимает разработанный слайдер фотографий. Он позволяет листать изображения в пределах одного плеера. Дополнительно реализована возможность скачивания просматриваемого фото на локальное устройство. За реализацию данного функционала отвечает метод `downloadPhoto`.

Метод `downloadPhoto` выполняет асинхронный запрос к фотографии по её URL-адресу и преобразует полученный ответ в бинарный формат. Далее через создание временной ссылки `<a>` происходит автоматическое инициирование скачивания в браузере. После происходит очистка временных ресурсов посредством вызова метода `URL.revokeObjectURL`.

```
async downloadPhoto() {
  try {
    const response = await fetch(this.fileURL);
    const blob = await response.blob();
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
```

					ПГУ1.090501.08.001 ПЗ	Лист
						77
Из	Лист	№ докум.	Подпись	Дата		

```

        a.download = this.dataForUserLayer.eventName || 'Без
названия';
        document.body.appendChild(a);
        a.click();
        document.body.removeChild(a);
        URL.revokeObjectURL(url);
    } catch (error) {
        errorHandlerNotify(error, "Скачивание фото");
    }
},

```

Дополнительно через панель действий пользователь может открыть панель просмотра атрибутивных данных события с метаданными.

Модуль интерфейса фотоплеера представлен на рисунке 20.

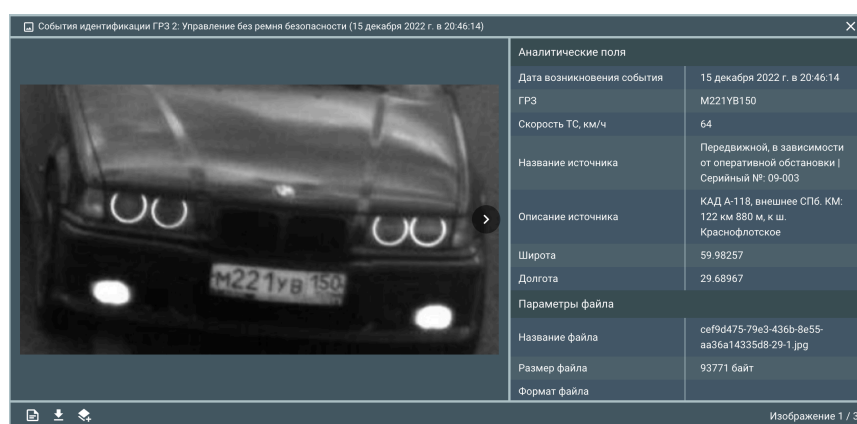


Рисунок 20 - Модуль интерфейса фотоплеера

5.8.2 Видеоплеер

Видеоплеер предназначен для воспроизведения потокового видео, предоставляемого источником данных. Имеется возможность просмотра прямой трансляции, а также архивной записи за определенный период времени.

Передача видеоданных осуществляется по технологии HLS. В данном подходе видеопоток разбивается на небольшие сегменты в формате .ts, каждый из которых соответствует определенному промежутку времени в 5-10 секунд. Сегменты собираются в плейлист формата .m3u8, который содержит информацию об их длительности, порядке воспроизведения и URL-адресе.

Для отображения видео на клиентской части по технологии HLS используется библиотека video.js. Внутри компонента плеера передается URL-адрес плейлиста. На основе него библиотека автоматически управляет сетевыми запросами на

получение сегментов видео.

За формирование ссылки на плейлист для просмотра прямого эфира отвечает метод `buildLiveStreamURL`. В качестве параметров она принимает уникальные идентификаторы источника и соответствующего набора. На основе них она возвращает сконструированный URL для плейлиста, пригодный для передачи внутрь плеера.

```
export const buildLiveStreamUrl = (sourceTypeId, sourceId) => {
  const API_URL = process.env.VUE_APP_API_URL;
  return `${API_URL}/object/${sourceTypeId}/${sourceId}/live/`;
}
```

За формирование ссылки на плейлист для архивного видеопотока отвечает функция `buildPeriodVideoURL`. Помимо уникальных идентификаторов источника и соответствующего набора она принимает даты начала и конца фрагмента видео.

```
export const buildPeriodVideoUrl = (sourceTypeId, sourceId, from,
to, formatToISO = true) => {
  const API_URL = process.env.VUE_APP_API_URL;
  const fromIso = formatToISO ? formatLocalISO(from) : from;
  const toIso = formatToISO ? formatLocalISO(to) : to;

  const filter = `NOT(('${fromIso}' > datetimeStart) OR ('${toIso}'
< datetimeStop))`;
  return
`${API_URL}/object/${sourceTypeId}/${sourceId}/m3u8/?where=${filter}`
}
```

Дополнительно, в пользовательском интерфейсе плеера отображается реальное календарное время текущего кадра. В названии файла каждого сегмента зашифрована дата и время начала соответствующего отрезка видео.

На этапе инициализации плеера на основе плейлиста строится временная шкала, в котором для каждого сегмента фиксируется начало и конец отрезка видео, которому он принадлежит, а также какой временной сдвиг необходимо провести, чтобы привести локальное время сегмента к реальному календарному времени. Во время воспроизведения видео система отслеживает текущее количество секунд от начала и на основе временной шкалы вычисляет реальное время кадра.

Модуль интерфейса видеоплеера представлен на рисунке 21.

					ПГУ1.090501.08.001 ПЗ	Лист
						79
Из	Лист	№ докум.	Подпись	Дата		

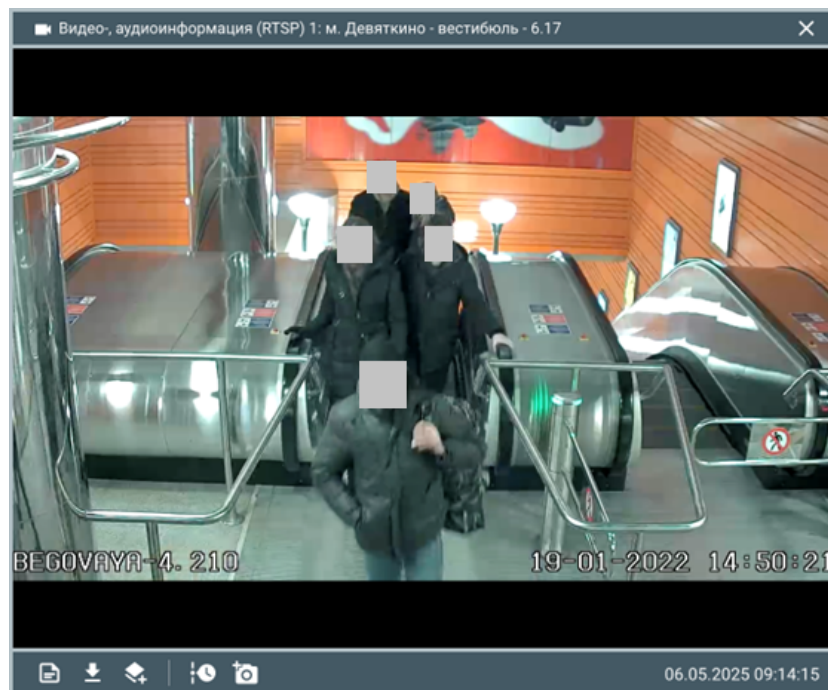


Рисунок 21 – Модуль интерфейса видеоплеера

Для возможности быстрого взаимодействия с режимом прямого эфира разработан интерактивный индикатор. Он интегрируется в интерфейс видеоплеера и имеет два вида отображения – для прямого эфира и для архивных записей.

В режиме прямого эфира индикатор автоматически отслеживает синхронизацию текущего кадра видео с концом видеопотока.

Он отображается в виде надписи «В ЭФИРЕ» с точкой слева. Если пользователь нажал на паузу или по каким-либо причинам отстал от эфира, цвет точки становится белым, если поток синхронизирован — красным, что представлено в соответствии с рисунками 22 и 23. При клике на индикатор плеер устанавливает текущий просматриваемый кадр в конец видеопотока, тем самым выполняя синхронизацию.



Рисунок 22 – Индикатор «В ЭФИРЕ» при синхронизированном воспроизведении (красная точка)

					ПГУ1.090501.08.001 ПЗ	Лист
Из	Лист	№ докум.	Подпись	Дата		80



Рисунок 23 – Индикатор «В ЭФИРЕ» при несинхронизированном воспроизведении (белая точка)

В режиме архивного просмотра вместо надписи «В ЭФИРЕ» отображается надпись «К ЭФИРУ» без индикаторной точки, что показано на рисунке 24. При нажатии на данный индикатор плеер переходит в режим прямой трансляции.

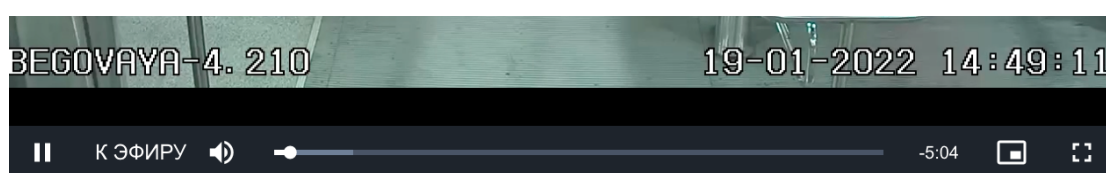


Рисунок 24 – Индикатор режима архивного просмотра «К ЭФИРУ»

Расширение реализовано как компонент библиотеки video.js и добавляется в нижнюю часть управления плеера справа от кнопки воспроизведения. Отслеживание состояния позиции кадра относительно конца видеопотока происходит путем подписки на события плеера timeupdate, loadedmetadata и pause, а также с помощью встроенного в библиотеку механизма liveTracker.

Доступ к видеопотоку осуществляется по технологии JWT аутентификации с использованием токена доступа. Так как он имеет свойство истекать через определенный период времени его необходимо своевременно обновлять. Для выполнения данной задачи разработано соответствующее расширение для видеоплеера. Оно выполняет функцию автоматического обновления токена доступа при получении от сервера ошибки с кодом 401 Unauthorized и обеспечивает непрерывное воспроизведение видеопотока при длительном просмотре избавляя пользователя от необходимости перезапуска плеера.

Разработанное расширение подключается к модулю VHS video.js плеера и устанавливает перехватчики сетевых запросов для сегментов видео и плейлистов. При получении ошибки с кодом 401 Unauthorized инициируется обновление токена

					ПГУ1.090501.08.001 ПЗ	Лист
						81
Из	Лист	№ докум.	Подпись	Дата		

доступа посредством вызова метода refreshToken сервисного класса AuthService. Если выполнение операции обновления прошло успешно старые перехватчики удаляются, источник видео устанавливается заново, воспроизведение видео продолжается, и обработчики сетевых запросов устанавливаются заново.

В случае возникновения ошибки плеер уведомляет об этом, выводя соответствующее сообщение.

При воспроизведении видеопотоков, предоставляемых различными внешними физическими устройствами, необходимо обрабатывать случаи поврежденных или некорректно закодированных сегментов видео. Они могут быть вызваны нестабильной сетью или некорректно настроенным оборудованием. В отличие от специализированных медиаплееров браузерные движки не способны отображать некорректно закодированные или поврежденные сегменты и отображение видеопотока полностью прерывается. Для обеспечения стабильной работы плеера при возникновении подобных случаев разработано соответствующее расширение. Его работа заключается в определении ошибки некорректной кодировки, в результате которого осуществляется автоматическая перемотка видео на следующий корректный сегмент.

Расширение подписывается на событие error плеера. В случае возникновения ошибки декодирования DECODE_ERROR запускается механизм восстановления воспроизведения. Плеер заново устанавливает URL-адрес источника видеопотока и выполняет перемотку вперед на 5 секунд с момента сбоя. Если после выполнения данной операции сегмент все ещё является некорректным, она повторяется.

					ПГУ1.090501.08.001 ПЗ	Лист
						82
Из	Лист	№ докум.	Подпись	Дата		

6 Развертывание и инфраструктура

6.1 Настройка веб-сервера Nginx

Для отображения пользовательского интерфейса при развертывании клиентской части на сервере, поддержки защищенного протокола HTTPS и взаимодействия с REST API используется веб-сервер nginx. В системе он выполняет роль обратного прокси и хостинга статических файлов.

Для реализации защищенного соединения подключаются SSL-сертификаты, протоколы и криптографические алгоритмы. Таким образом организуется безопасная передача данных между клиентом и сервером.

```
listen      443 ssl;
listen      [::]:443 ssl;
server_name localhost;
ssl_certificate /etc/nginx/ssl/platform.crt;
ssl_certificate_key /etc/nginx/ssl/device.key;
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers HIGH:!aNULL:!MD5;
```

Сетевые запросы от клиента, начинающиеся с префикса /api/v2/ автоматически перенаправляются на соответствующий адрес сервера REST API, переданный в переменные окружения Docker-контейнера.

```
location /api/v2/ {
    proxy_pass __API_URL__;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
}
```

6.2 Контейнеризация приложения

Для удобства развертывания и масштабирования веб-приложение была контейнеризировано с использованием Docker.

Это позволяет собрать его один раз и развертывать в любом окружении независимо от операционной системы и инфраструктуры.

Сборка контейнера с веб-приложением осуществляется в несколько этапов через Dockerfile. Вначале происходит сборка клиентской части путем загрузки

					ПГУ1.090501.08.001 ПЗ	Лист
						83
Из	Лист	№ докум.	Подпись	Дата		

образа Node.js, установки зависимостей проекта и самой сборки приложения. В результате собранное приложение помещается в директорию dist контейнера.

```
FROM node:lts-alpine as build-stage
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
```

На втором этапе происходит развертывание приложения через образ nginx. В него копируются статические файлы из директории dist, шаблон конфигурации, SSL-сертификаты и скрипт инициализации entrypoint.sh, автоматически выполняющийся при запуске контейнера.

```
FROM nginx:stable-alpine as production-stage
COPY --from=build-stage /app/dist /usr/share/nginx/html
COPY ./nginx/nginx.conf.template /etc/nginx/nginx.conf.template
COPY ./nginx/key-ssl /etc/nginx/ssl
COPY ./nginx/ssl.conf.template /etc/nginx/ssl.conf.template
COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh
ENTRYPOINT ["/entrypoint.sh"]
```

Ключевой особенностью реализации является возможность передачи посредством переменных окружения URL-адреса REST API при запуске контейнера без необходимости пересборки образа. Это позволяет использовать один и тот же образ в разных окружениях. Данный механизм реализуется через скрипт entrypoint.sh. При запуске контейнера происходит автоматическая замена плейсхолдера __API_URL__ на значение переменной окружения API_URL.

```
cp /etc/nginx/nginx.conf.template /etc/nginx/nginx.conf
sed -i "s|__API_URL__|${API_URL}|g" /etc/nginx/nginx.conf
echo "Starting Nginx..."
exec nginx -g 'daemon off;'
```

					ПГУ1.090501.08.001 ПЗ	Лист
						84
Из	Лист	№ докум.	Подпись	Дата		

7 Тестирование

Для оценки корректности работы разработанного клиентского приложения было проведено функциональное тестирование основных компонентов системы. Тестирование направлено на проверку выполнения ключевых пользовательских сценариев и устойчивости интерфейса при различных условиях эксплуатации.

Результаты тестирования приведены в таблице 3.

Таблица 3 – Результаты тестирования клиентской части геоинформационной платформы

Тестируемая функция	Входные данные	Ожидаемый результат	Результат
Авторизация: Локальная	Ввод логина и пароля	Пользователь перенаправлен на /main	Пройдено
Авторизация: Keycloak	Ввод данных на стороне Keycloak	Возврат в систему, переход на /main	Пройдено
Обновление токена	Искусственное истечение токена	Токен обновлен, сессия продолжается	Пройдено
Защита маршрутов	Попытка перехода на /main без авторизации	Переход на /login	Пройдено
Кеш иконок	Перезагрузка страницы	Иконки не загружаются заново	Пройдено
Кластеризация	Перезагрузка страницы	Источники группируются корректно	Пройдено
Отображение подвижных объектов	Подвижные наборы источников получены	Графическое обозначение отображается	Пройдено
Показ маршрута	Выбран необходимый маршрут	Отображается линия с направлением	Пройдено
Фильтрация по области	Выделение полигона	Список источников обновляется	Пройдено
Фотоплеер	Просмотр и скачивание фото	Фото скачивается, отображается	Пройдено

Тестируемая функция	Входные данные	Ожидаемый результат	Результат
Видеоплеер: Live	Открытие live-потока	Поток отображается	Пройдено
Видеоплеер: Архив	Выбор интервала	Отображается архивное видео	Пройдено
Обновление токена в плеере	Live-поток, токен истек	Происходит обновление токена	Пройдено
Ошибка декодирования	Поврежденный сегмент	Перемотка на 5 сек вперед	Пройдено

На основании проведенного тестирования можно сделать вывод, что приложение выполняет все заявленные функции в соответствии с техническим заданием. Основные пользовательские сценарии отрабатываются корректно, критических сбоев не зафиксировано. Обработка исключительных ситуаций реализована корректно, устойчивость системы при длительной работе подтверждена. Таким образом, клиентская часть геоинформационной платформы может быть рекомендована к внедрению в эксплуатацию.

8 Оценка затрат на разработку и внедрение

8.1 Организация процесса разработки

Разработка клиентской части геоинформационной платформы сбора данных с интеграцией медиаконтента осуществлялась в индивидуальном порядке. В качестве рабочего места разработка использовалось арендованное помещение общей площадью 15 м², при ориентировочной стоимости его аренды в целом 12000 рублей в месяц.

Для осуществления процесса разработки использовалась современная вычислительная техника, соответствующее программное обеспечение и дополнительные материалы. Основная часть программных средств представлена в виде открытого и свободного ПО, однако также применялись лицензионные комплекты.

Данные по потребности в оборудовании и лицензионном программном обеспечении представлены в соответствующих таблицах 4 и 5.

Таблица 4 – Потребность в оборудовании

Оборудование	Источник	Количество, шт.	Ориентировочная цена за единицу, руб.	Потребляемая мощность, Вт
Ноутбук Huawei MateBook D15, 15.6"	ООО «Ситилинк»	3	52 990	65
Монитор Philips 24E1N3300A/01, 23.8"	ООО «DNS»	3	9 990	24
Клавиатура Logitech K280e	ООО «OZON»	3	2 290	3
Мышь Logitech M110 Silent	ООО «OZON»	3	890	2
Источник бесперебойного питания APC BX500MI	ООО «Ситилинк»	3	5 490	300

Таблица 5 – Потребность в лицензионном программном обеспечении

Лицензионное программное обеспечение	Источник	Цена, руб.	Срок действия лицензии, дней
Linux Ubuntu 22.04	Официальный сайт Ubuntu:	-	бессрочная

Лицензионное программное обеспечение	Источник	Цена, руб.	Срок действия лицензии, дней
	https://ubuntu.com/download		
Visual Studio Code	Официальный сайт Visual Studio Code: https://code.visualstudio.com/	-	бессрочная
Антивирус Kaspersky Plus 2023	Интернет-магазин Kaspersky: https://www.kaspersky.ru/	1990	бессрочная
Node.js	Официальный сайт Node.js: https://nodejs.org	-	бессрочная
Docker	Официальный сайт Docker: https://www.docker.com/	-	бессрочная

Перечень дополнительных материалов, источники их приобретения, объем, и ориентировочные цены приведены в таблице 6.

Таблица 6 – Потребность в дополнительных материалах

Материал	Источник	Количество, штук	Ориентировочная цена за единицу, руб.
Компьютерный стол Simple 110×75×60 см, ЛДСП, белый	ООО «Леруа Мерлен Восток»	3	2800
Светодиодная настольная лампа СН-360, 10 Вт, диммер	ООО «Леруа Мерлен Восток»	3	3900
Офисное кресло Ridberg Ch-695, ткань, серый цвет	ООО «Леруа Мерлен Восток»	3	4200
Канцелярские принадлежности	Гипермаркет «Канцлер»	3 комплекта	1000

Разработка клиентского приложения геоинформационной платформы осуществлялась командой разработчиков, представленной в таблице 7.

Таблица 7 – Состав коллектива разработчиков

Специалист	Количество, чел.	Описание занятости
Руководитель проекта	1	Управление проектом. Планирование этапов разработки. Консультирование по архитектурным решениям.

Специалист	Количество, чел.	Описание занятости
		Контроль выполнения требований. Приёмка результата.
Программист	1	Разработка клиентской части веб-приложения. Реализация визуализации геоисточников и маршрутов. Интеграция с REST API. Оптимизация производительности.
Тестировщик / UI-инженер	1	Тестирование визуальных компонентов. Проверка отображения данных на карте. Тестирование авторизации и взаимодействия с сервером. Анализ UX и UI.

Данные по основной заработной плате разработчиков клиентской части геоинформационной платформы и их количественному составу приведены в таблице 8.

Таблица 8 – Сведения о заработной плате

Специалист	Количество, чел.	Сумма заработной платы в месяц, руб.
Руководитель проекта	1	100000
Программист	1	80000
Тестировщик / UI-инженер	1	50000

8.2 Расчет себестоимости программного обеспечения

Себестоимость разработки клиентской части геоинформационной платформы сбора данных включает следующие основные статьи расходов:

- приобретение оборудования;
- приобретение лицензионного программного обеспечения;
- приобретение дополнительных материалов;
- заработная плата членов команды;
- отчисления на страховые взносы;
- обслуживание программы на этапе разработки.

Затраты на оборудование в период разработки составили 214950 рублей и представлены в таблице 4.

Затраты на программное обеспечение в период разработки составили 1990 рублей и представлены в таблице 5.

Затраты на дополнительные материалы в период разработки составили 33700 рублей и представлены в таблице 6.

Затраты на заработную плату включают выплаты сотрудникам и обязательные страховые отчисления.

Количество рабочих дней в среднем составляет 21 день в месяц. Продолжительность разработки проекта составила 3 месяца.

Суммарная месячная заработная плата всей команды составила 230000 рублей, соответственно, за весь период разработки 690000 рублей.

Страховые взносы начисляются в соответствии с действующим законодательством РФ и включают:

- в Пенсионный фонд РФ (22%);
- в Фонд обязательного медицинского страхования (5,1%);
- в Фонд социального страхования РФ (2,9%).

Отчисления на страховые взносы производятся в соответствии с Налоговым кодексом Российской Федерации от 14.07.2022 № 239-ФЗ [8]. С 2023 года для организаций, осуществляющих деятельность в области информационных технологий и соответствующих требованиям, установлен единый пониженный тариф страховых взносов — 7,6% от суммы начисленной заработной платы.

Таким образом, страховые взносы рассчитываются следующим образом: 30% от суммы в пределах МРОТ и 7,6% от суммы, превышающей МРОТ. МРОТ на 2025 год составляет 22440 рублей в месяц.

Данные по суммам отчислений на страховые взносы приведены в таблице 9.

					ПГУ1.090501.08.001 ПЗ	Лист
						90
Из	Лист	№ докум.	Подпись	Дата		

Таблица 9 – Отчисления на страховые взносы

Сотрудник	Заработная плата, руб.	Взнос по единой ставке, руб.	Сумма, подлежащая пониженной ставке по взносу, руб.	Взнос по пониженной ставке, руб.	Общая сумма взноса с сотрудника за месяц, руб.
Руководитель проекта	100000	6732	77560	5894,6	12626,56
Программист	80000	6732	57560	4374,6	11106,56
Тестировщик	50000	6732	27560	2094,6	8826,56

Затраты на заработную плату и отчисления на страховые взносы представлены в таблице 10.

Таблица 10 – Затраты на заработную плату

Сотрудник	Заработная плата, руб.	Общая сумма отчислений, руб.	Всего за 1 месяц, руб.	Всего за 3 месяца, руб.
Руководитель проекта	100000	12626,56	112626,56	337879,68
Программист	80000	11106,56	91106,56	273319,68
Тестировщик	50000	8826,56	58826,56	176479,68

Затраты на заработную плату команды проекта составляют 787571,4 руб., из которых 690000 руб. – основная заработная плата, а 97679,04 руб. – сумма обязательных страховых отчислений.

Затраты на обслуживание включают следующие статьи:

- амортизация оборудования, составляющая 20% от его стоимости;
- обслуживание оборудования, составляющее 15% от суммы амортизации;
- затраты на электроэнергию, включающие затраты на питание вычислительной техники и освещения;
- расходы на содержание помещения, включающие арендные и эксплуатационные платежи;
- накладные расходы.

Для реализации проекта были приобретены ноутбуки, мониторы и периферия

на общую сумму 214 950 руб.

Таким образом, общая сумма затрат на оборудование составила 214950 рублей. Амортизация оборудования принята в размере 20 % в год, что эквивалентно ежемесячным амортизационным отчислениям в 3582,5 рубля.

Затраты на обслуживание вычислительной техники оцениваются как 15 % от амортизационных отчислений, что составляет 537,38 рубля в месяц.

Электроэнергия расходуется при работе на трёх ноутбуках мощностью 65 Вт каждый в течение 8 часов ежедневно. За весь период эксплуатации (63 дня) общее потребление энергии составило около 98,28 кВт·ч. Дополнительно учитывается 25 кВт·ч на периферийное оборудование. В сумме это составляет 123,28 кВт·ч. При средней стоимости электроэнергии 6,5 руб/кВт·ч затраты составляют примерно 801,32 рубля.

Накладные расходы рассчитываются как 75 % от фонда оплаты труда, равного 750 821,20 рубля, и составляют 563 115,90 рубля.

Результат расчетов по статьям затрат на обслуживание приведены в таблице 11.

Таблица 11 – Затраты на обслуживание

Статья затрат	Показатель отчислений	Сумма отчислений в месяц, руб.	Итоговая сумма отчислений за период разработки, руб.
Амортизация оборудования	20% от стоимости в год	3582,5	10747,5
Обслуживание оборудования	15% от амортизации	537,38	1612,14
Затраты на электроэнергию	6,5 руб./кВт·ч	-	972,97
Аренда офиса и расходы на его содержание	-	15 000	36000
Накладные расходы	75% от фонда оплаты труда	-	590759,28
Итого			640093,75

Себестоимость разработки определяется как сумма всех затрат, включающих основные статьи расходов, необходимые для реализации проекта. Сводная информация представлена в таблице 12.

Таблица 12 – Общая сумма затрат

Статья затрат	Сумма, руб.
Приобретение оборудования	214950
Приобретение лицензионного программного обеспечения	1990
Затраты на заработную плату	690000
Отчисления на страховые взносы	97679,04
Затраты на дополнительные материалы	33700
Затраты на обслуживание	640093,75
Итого	1678412,79

Таким образом, сумма затрат на разработку проекта составила 1655828,81 рублей и является себестоимостью разработки клиентской части геоинформационной системы сбора данных с интеграцией медиаконтента.

8.3 Оценка эффективности разработки

Оценка экономической и функциональной эффективности определяет необходимость и целесообразность разработки системы. Экономическая эффективность инвестиционных проектов характеризуется системой показателей, отражающих соотношение результатов (притока, поступлений, дохода) и затрат (оттока, выплат) от операционной, инвестиционной и финансовой деятельности.

Внедрение клиентской части геоинформационной системы сбора данных с интеграцией медиаконтента обеспечивает не только технические и организационные преимущества, но и дает неисчисляемые эффекты, влияющие на повышение производительности и надежности работы всей информационной инфраструктуры.

К таким эффектам можно отнести:

- повышение оперативности принятия решений;
- упрощение доступа к мультимедийному контенту, предоставляемому физическими источниками;
- снижение нагрузки на персонал за счет автоматизации и визуализации процессов;
- обеспечение технологического суверенитета за счет отсутствия зависимости от сторонних или зарубежных сервисов.

8.4 Анализ рисков

В процессе проектирования и разработки геоинформационной платформы могут возникнуть различного рода риски, влияющие на качество и результативность выполняемого проекта.

Недостаточная коммуникация между сторонами может привести к недопониманиям или неверному толкованию задач, целей и сроков. Для минимизации данных рисков необходимо заранее определить ответственных лиц со всех сторон, регламентировать периодичность встреч и вести постоянную фиксацию промежуточных результатов и требований.

Функциональные риски, возникающие при несоответствии функционала системы заявленным требованиям, могут быть следствием неточного технического задания или неуточненных деталей реализации. Для их снижения был проведен подробный анализ предметной области, спроектирована схема взаимодействия элементов интерфейса, проведено модульное тестирование.

Уход из команды разработчика с уникальными профессиональными знаниями может затормозить работу. Мерами снижения риска стало использование системы контроля версий, а также регулярная передача знаний внутри команды разработчиков проекта.

Использование нелицензионного ПО может привести к штрафам. Для реализации проекта применялись исключительно свободно распространяемые или лицензионные программные продукты.

Выявленные при предпроектном анализе возможные риски и пути их минимизации позволили обеспечить стабильность разработки, своевременное завершение проекта и готовность системы к внедрению в инфраструктуру заказчика.

					ПГУ1.090501.08.001 ПЗ	Лист
						94
Из	Лист	№ докум.	Подпись	Дата		

9 Обеспечение безопасности труда при разработке

9.1 Актуальность вопросов безопасности

В условиях активного развития информационных технологий и цифровизации процессов персональные вычислительные средства стали неотъемлемой частью жизнедеятельности человека. В том числе это относится и к геоинформационным системам, где оператор взаимодействует с большими наборами данных и ведет аналитическую деятельность.

Особенности эксплуатации платформы предполагает длительное нахождение за персональным компьютером и работу с визуальным нагруженным пользовательским интерфейсом. Это требует постоянной концентрации внимания для взаимодействия с множеством интерактивных элементов. Вопросы обеспечения безопасности жизнедеятельности являются особенно актуальными для сохранения здоровья и поддержания высокой работоспособности оператора.

Согласно ГОСТ 12.0.003-74, вредные производственные факторы делятся на 4 категории:

- физические – недостаточная или избыточная освещенность, яркий свет экрана монитора, неподходящая температура или влажность воздуха помещения, повышенный уровень шума;
- психофизиологические – продолжительное статичное нахождение тела, нагрузка на зрение при чтении с экрана монитора, перегрузка нервной системы при работе с большим объемом данных;
- биологические – риск заражения вирусными и бактериальными инфекциями при работе в помещениях с постоянным скоплением людей;
- химические – воздействие вредных веществ, находящихся в воздухе.

9.2 Техника безопасности при работе с электроприборами

В процессе эксплуатации клиентской части геоинформационной платформы сбора данных предполагается использование операторами персональных компьютеров и периферийных устройств, относящихся к категории

					ПГУ1.090501.08.001 ПЗ	Лист
						95
Из	Лист	№ докум.	Подпись	Дата		

электроприборов. Необходимо строго соблюдать технику безопасности:

- все подключаемое оборудование, включая розетки, должно быть заземлено;
- запрещается использование электрооборудования во влажной среде;
- отсоединение электрооборудования от сети должно осуществляться касанием только через вилку;
- перед подключением электрооборудования необходимо проверить целостность питающего кабеля.

При работе с техникой необходимо обеспечивать соблюдение норм расстояния, вентиляции и ограничения непрерывного времени работы.

9.3 Влияние ПЭВМ на физиологию человека

Работа оператора с клиентской частью геоинформационной платформы осуществляется за персональным компьютером и предполагает длительное взаимодействие с интерфейсом. Такой вид труда оказывает значительную нагрузку на зрительный аппарат, нервную систему и опорно-двигательный аппарат человека.

При продолжительной работе могут наблюдаться снижение остроты зрения, утомление и снижение концентрации внимания, а также ухудшение кровообращения из-за длительного нахождения в неподвижной позе.

Необходимо учитывать следующие факторы риска:

- несоответствие стола и кресла анатомическим особенностям человека;
- неправильный угол наклона спины и головы;
- некорректно установленный монитор;
- длительное отсутствие перерывов в работе.

Согласно СанПиН 2.2.3670-20 «Санитарно-эпидемиологические требования к условиям труда», допустимый уровень шума не должен превышать 50 дБА. Превышение данного значения может привести к снижению умственной работоспособности, головной боли и раздражительности.

Для минимизации акустических нагрузок применяются шумоизолирующие материалы в помещении и ограничение посторонних звуков в рабочей зоне.

					ПГУ1.090501.08.001 ПЗ	Лист
						96
Из	Лист	№ докум.	Подпись	Дата		

9.4 Техника безопасности при работе с ПК

На основе типовой инструкции по охране труда при работе на ПК (персональном компьютере) ТОО Р-45-084-01 [10], которая была утверждена Приказом Минсвязи Российской Федерации №162 от 02.07.2001, деятельность на ЭВМ делится на 3 группы:

- группа А – чтение информации с экрана. Рекомендуется делать два 15-минутных перерыва в течение рабочей смены (не включая обед);
 - группа Б – набор текстовой информации. Рекомендуются 10-минутные перерывы после каждого часа работы;
 - группа В – интеллектуально-напряжённая или творческая деятельность.
- Предусматриваются перерывы по 15 минут каждый час.

Рекомендуется использовать только сертифицированную и исправную вычислительную технику, обеспечивать ее устойчивое положение. С точки зрения физиологии человека необходимо делать регулярные перерывы с проведением гимнастики для глаз, разминок мышц шей и спины.

					ПГУ1.090501.08.001 ПЗ	Лист
						97
Из	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе выполнения дипломной работы была достигнута основная цель — проектирование и разработка клиентской части геоинформационной платформы, предназначенной для отображения физических источников и визуальной оценки медиаданных, связанных с ними. Проведён анализ предметной области, изучены существующие подходы и решения, определены ключевые функциональные и технические требования к системе.

В процессе реализации клиентской части были решены следующие задачи:

- спроектирована архитектура клиентской части;
- реализован механизм пользовательской авторизации и аутентификации с поддержкой технологии JWT и интеграцией с внешним провайдером Keycloak;
- обеспечено получение, отображение и кеширование информации о наборах источников и их иконках;
- разработан модуль визуализации статичных и подвижных источников с поддержкой кластеризации, маршрутов и отображения на карте;
- реализован механизм фильтрации источников по выделенной пользователем области;
- внедрены медиаплееры для просмотра фото- и видеоконтента, включая обработку ошибок при воспроизведении потоков;
- выполнена настройка веб-сервера Nginx и контейнеризация приложения с использованием Docker.

Разработанная система успешно прошла тестирование, подтверждающее её работоспособность, корректность отображения данных и устойчивость к ошибкам. Клиентская часть платформы адаптирована для работы в локальной инфраструктуре без доступа к внешним сервисам, что делает её применимой в задачах специального и ведомственного назначения.

Практическая значимость работы заключается в том, что предложенное решение может быть использовано в системах мониторинга, инфраструктурного управления, обеспечения безопасности и логистики.

					ПГУ1.090501.08.001 ПЗ	Лист
						98
Из	Лист	№ докум.	Подпись	Дата		

В перспективе возможна интеграция дополнительных типов наборов источников, расширение функционала медиаплееров и реализация аналитических модулей на стороне клиента.

					ПГУ1.090501.08.001 ПЗ	Лист
						99
Из	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Coughlin T. 175 Zettabytes By 2025 [Электронный ресурс]/Режим доступа: <https://www.forbes.com/sites/tomcoughlin/2018/11/27/175-zettabytes-by-2025/>, свободный. (дата обращения: 13.05.2025).
2. Esri. Решения и технологии в сфере общественной безопасности: программное ГИС-обеспечение для картографирования [Электронный ресурс]/Режим доступа: <https://www.esri.com/ru-ru/industries/public-safety/overview>, свободный. (дата обращения: 10.05.2025).
3. Совзонд. Роль геопространственных систем в повышении эффективности и рентабельности городских коммунальных служб [Электронный ресурс]/Режим доступа: <https://sovzond.ru/press-center/news/corporate/7704/>, свободный. (дата обращения: 09.05.2025).
4. Сбербанк. ГИС для бизнеса: GeoAI, IoT для городов, большие данные с геотегами и геопространственное мышление [Электронный ресурс]/Режим доступа: <https://habr.com/ru/companies/sberbank/articles/589817/>, свободный. (дата обращения: 07.05.2025).
5. NextGIS. Официальный сайт [Электронный ресурс]/Режим доступа: <https://nextgis.ru/>, свободный. (дата обращения: 05.05.2025).
6. GeoMixer. Веб-геоинформационная платформа для широкого спектра задач [Электронный ресурс]/Режим доступа: <http://geomixer.ru/>, свободный. (дата обращения: 03.05.2025).
7. ГИС ПАНОРАМА. GIS WebServer Special Edition (SE) [Электронный ресурс]/Режим доступа: <https://gisinfo.ru/products/giswebserverse.htm>, свободный. (дата обращения: 28.04.2025).
8. Wialon. Управление и контроль автопарка в реальном времени [Электронный ресурс]/Режим доступа: <https://wialon.com/ru>, свободный. (дата обращения: 21.04.2025).
9. CNews. Ситуационные центры: для безопасности и не только [Электронный ресурс]/Режим доступа: https://www.cnews.ru/reviews/it_v_

					ПГУ1.090501.08.001 ПЗ	Лист
						100
Из	Лист	№ докум.	Подпись	Дата		

organah_gosudarstvennoj_vlasti_2013/articles/situatsionnye_tsentry_dlya_bezopasnosti
_i_ne_tolko, свободный. (дата обращения: 14.04.2025).

					ПГУ1.090501.08.001 ПЗ	Лист
						101
Из	Лист	№ докум.	Подпись	Дата		

Приложение А

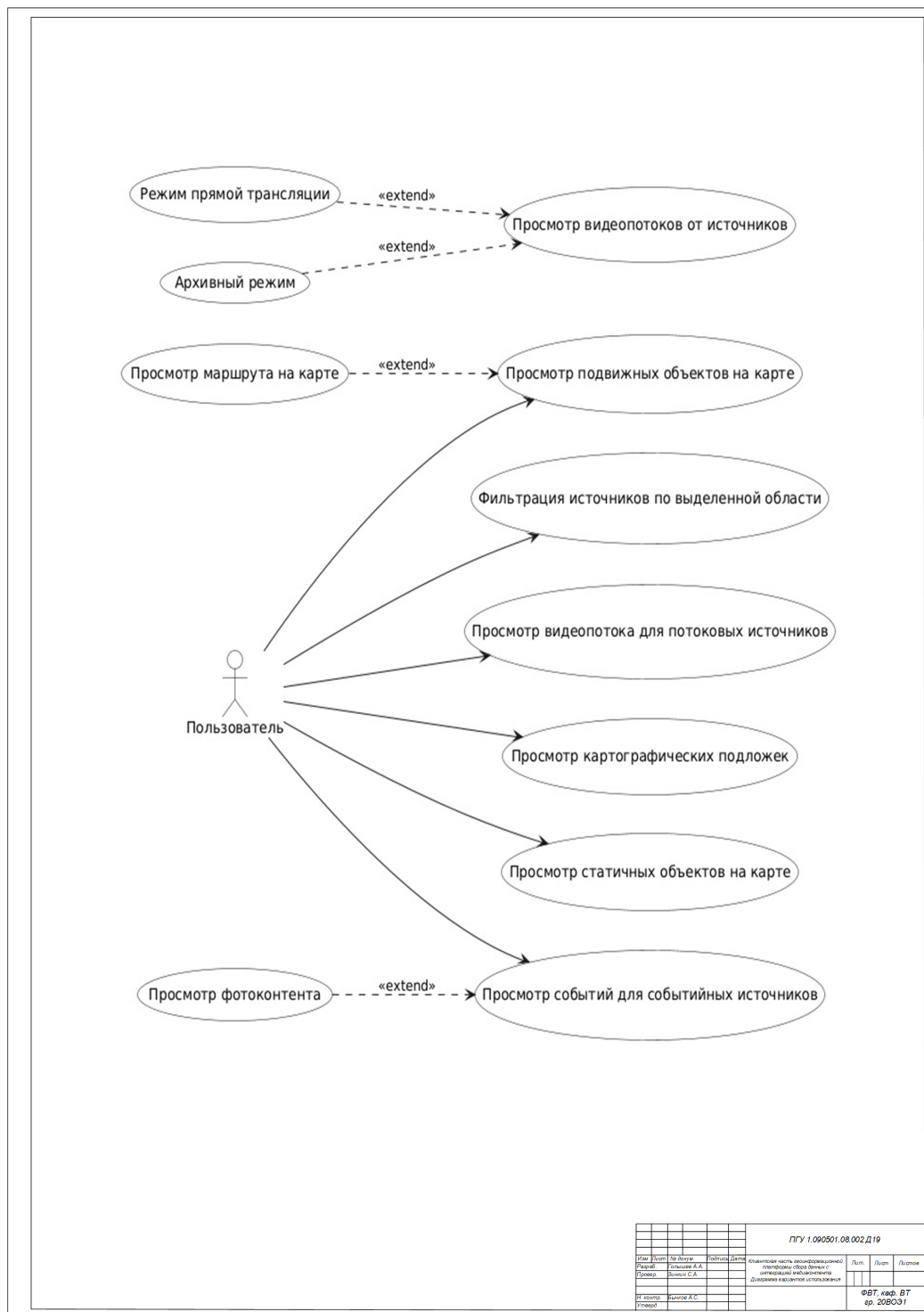
(обязательное)

Клиентская часть геоинформационной платформы сбора данных

UML-диаграммы

					ПГУ1.090501.08.001 ПЗ	Лист
						102
Из	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – Диаграмма вариантов использования



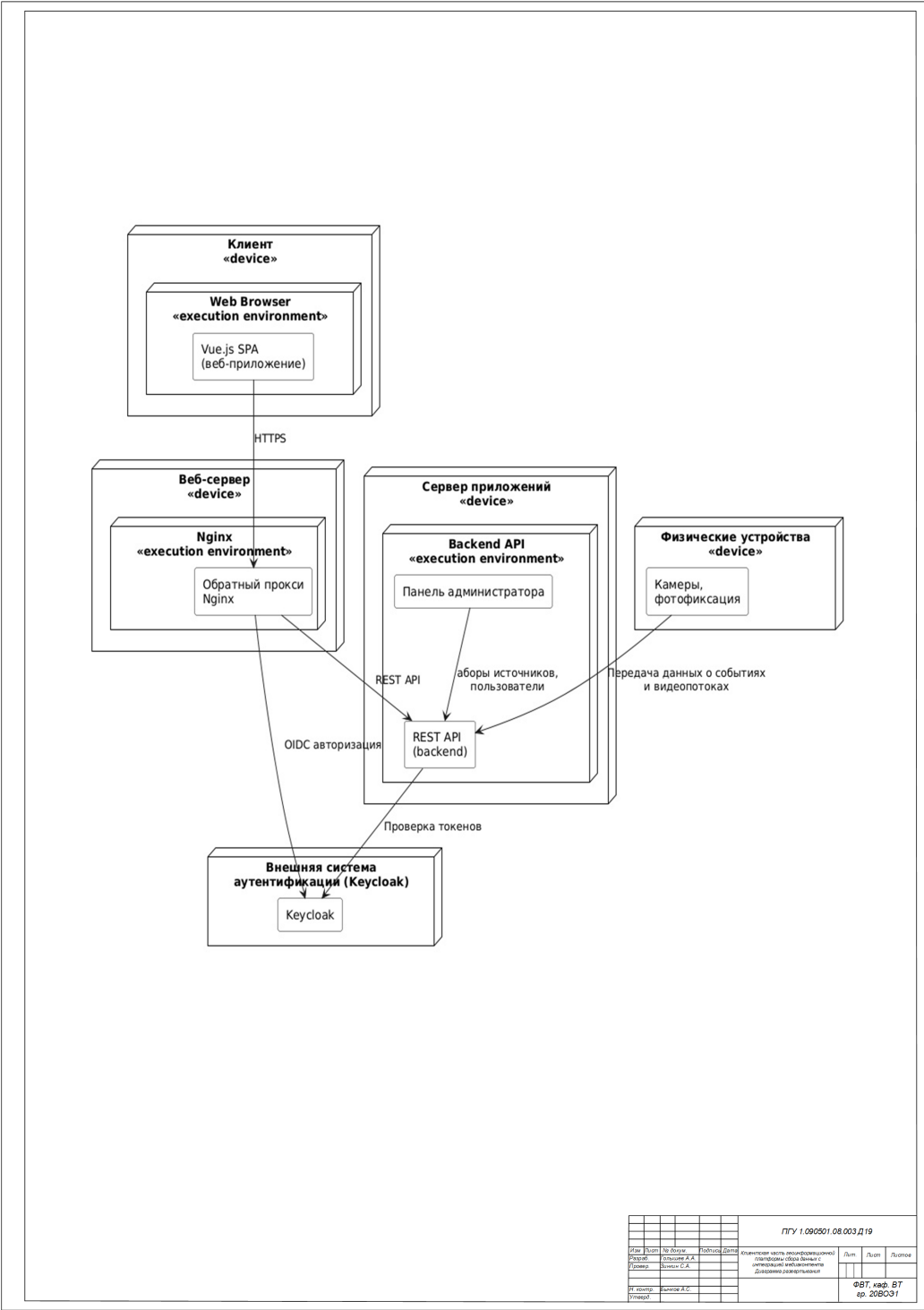
Из	Лист	№ докум.	Подпись	Дата

ПГУ1.090501.08.001 ПЗ

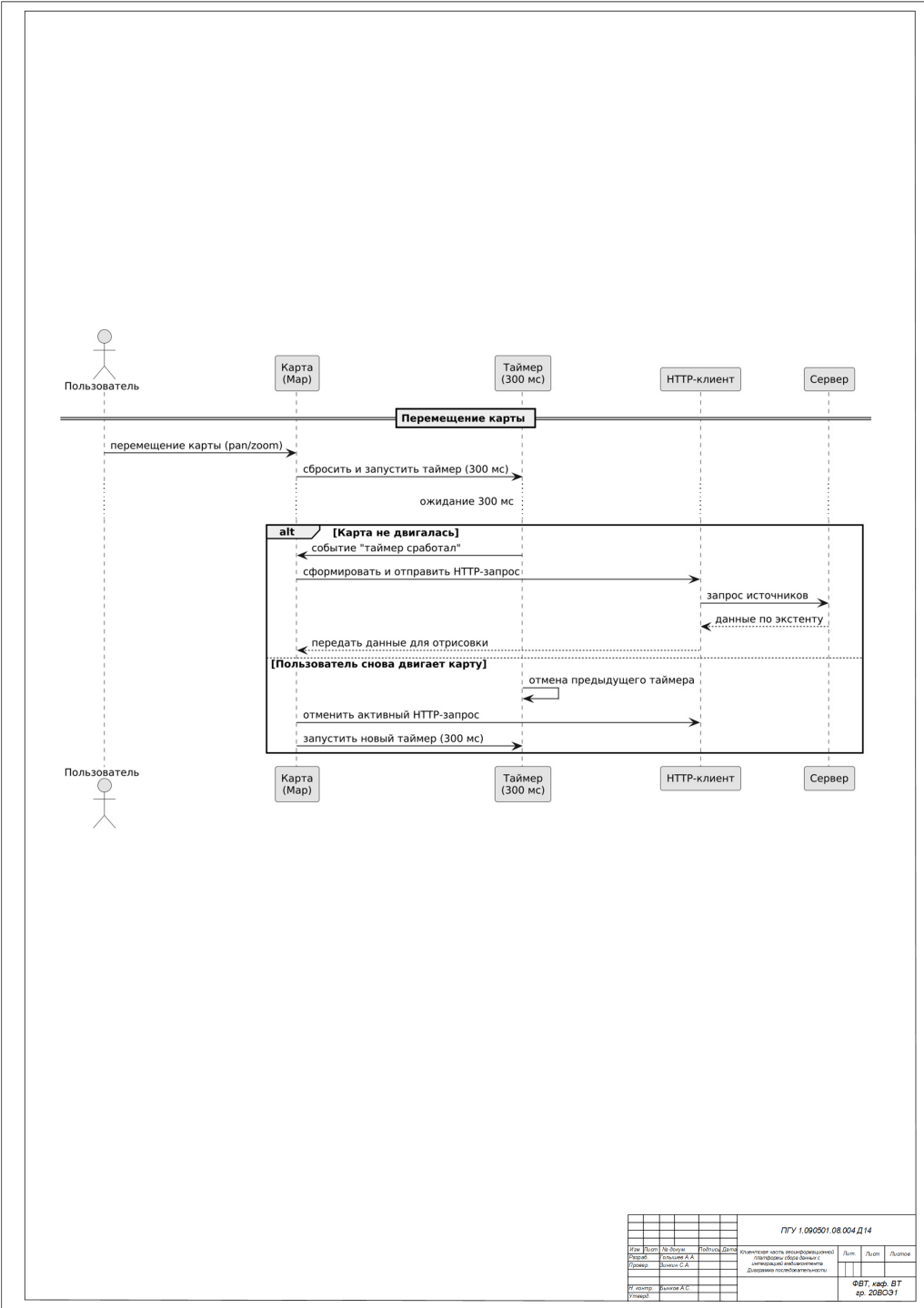
Лист

103

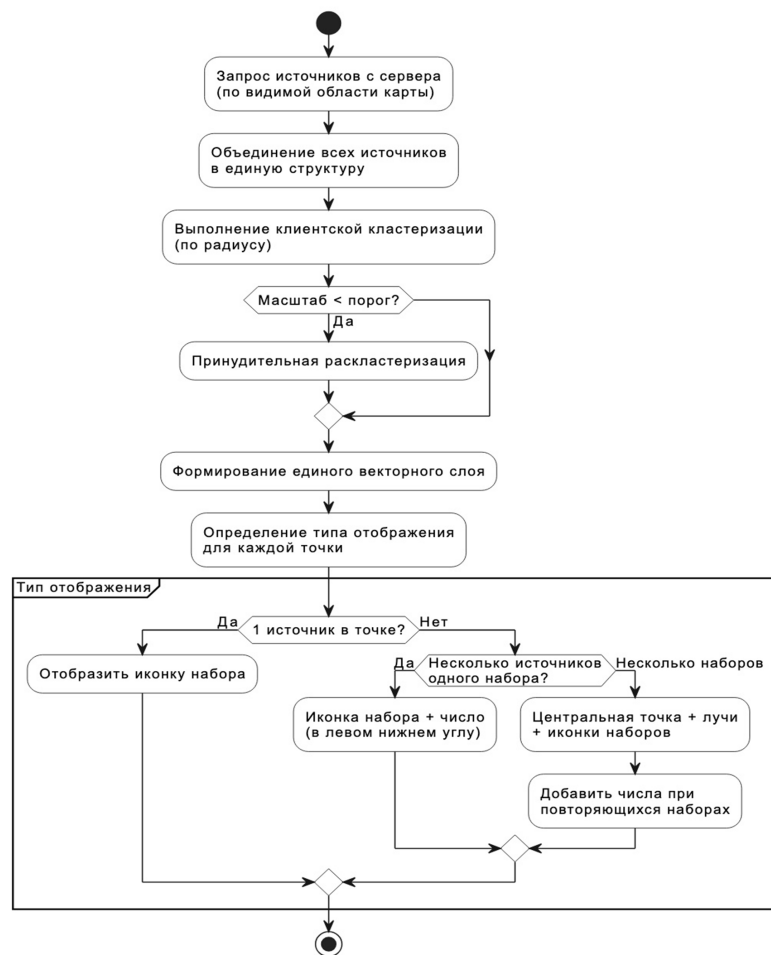
Приложение А.2 – Диаграмма развертывания



Приложение А.3 – Диаграмма последовательности

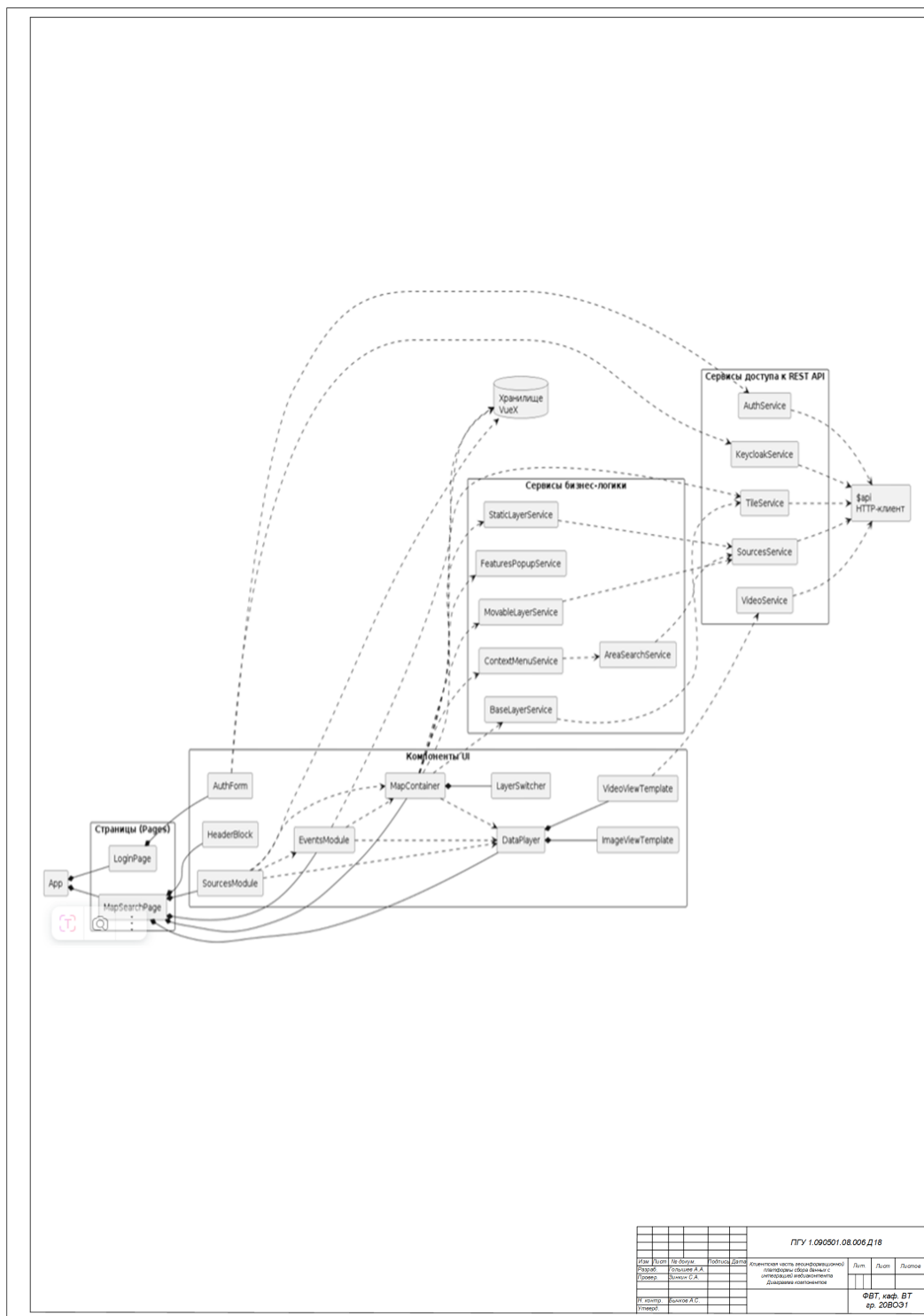


Приложение А.4 – Диаграмма деятельности



						ПГУ 1.090501.08.005 Д17						
Имя	Фамилия	Полное имя	Подпись	Дата	Клиентская часть (информационной платформы сбора данных с интегрированными механизмами выявления деятельности)					Лист	Лист	Листов
Рязань	Полынин А.А.											
Гровер	Зинкин С.А.											
Имя	Фамилия	Полное имя	Подпись	Дата								
Углов	Балов А.С.											
										ФВТ, каф. ВТ зр. 20В031		

Приложение А.5 – Диаграмма компонентов



Приложение Б

(Обязательное)

Клиентская часть геоинформационной платформы сбора данных

Листинг программы

					ПГУ1.090501.08.001 ПЗ	Лист
						108
Из	Лист	№ докум.	Подпись	Дата		

Файл StaticLayerService.js

```
import Vue from 'vue';
import VectorLayer from 'ol/layer/Vector';
import VectorSource from 'ol/source/Vector';

import {StaticLayerStyles} from
'@/services/StaticLayerService/StaticLayerStyles';
import {
  getExtentForMapView,
  getFeaturesFromGeoJSON,
  styleHandler,
} from '@/services/StaticLayerService/StaticLayerUtils';
import {debounce} from '@/utils/debounce';
import {sharedData} from '@/utils/shared-data';
import {TileService} from '@/api/services/TileService';
import clusterFeatures from '@/utils/clusterFeatures';
import {EventBus} from '@/utils/bus';
import {getFeatureDataForPixel} from '@/utils/mapUtils';
export class StaticLayerService {
  constructor(sourceTypesInfo, sourcesType) {
    this.sourceTypesInfo = Vue.observable(sourceTypesInfo.map(info
=> ({
      ...info,
      visibility: info?.defaultVisibility ?? true
    })));
    this.sourcesType = sourcesType;
    this.styleService = new StaticLayerStyles(sourceTypesInfo);
    this.abortController = new AbortController();
    this.map = undefined;
    this.extent = undefined;
    this.layer = new VectorLayer({style: (f) => styleHandler(f,
this.styleService, this.map)});
    this.updateIntervalsId = new Map();
    this.lastFeaturesCount = {};
    this.setUpdateHandlers();
  }
  clearIntervals() {
    this.updateIntervalsId.forEach(intervalId =>
clearInterval(intervalId));
  }
  getEventsHandlers() {
    return [
      this.getMoveMapHandler(),
      this.getShowPlayerHandler()
    ]
  }
  getMoveMapHandler() {
    const handleExtentUpdate = debounce(async (event) => {
      this.map = event.map;
      this.extent = getExtentForMapView(this.map);

      await this.updateFeaturesOnMap();
    }, 300)
    return {
      key: 'moveend',
      action: (event) => handleExtentUpdate(event)
    }
  }
}
```

					ПГУ1.090501.08.001 ПЗ	Лист
						109
Из	Лист	№ докум.	Подпись	Дата		

```

    }
  }
  getShowPlayerHandler() {
    const handleShowPlayer = (e) => {
      const {feature} = getFeatureDataForPixel(this.map,
e.pixel);
      if (!feature) return;
      const sourceTypeId = feature.get('sourceType');
      const sourceTypeInfo = this.sourceTypesInfo.find(info =>
info.sourceType === sourceTypeId);
      if (!sourceTypeInfo) return;
      const isStatic = sourceTypeInfo.sourceTypeView === 0 ||
sourceTypeInfo.sourceTypeView === 1 || sourceTypeInfo.sourceTypeView === -
1;

      if (!isStatic) return;
      const properties = feature.getProperties();
      const {ids, names, descriptions, sourceType} = properties;
      if (ids.length > 1) {
        EventBus.$emit('onCustomViewSources', {feature});
      } else {
        if (sourceTypeInfo.sourceTypeView === 1 ||
sourceTypeInfo.sourceTypeView === -1) {
          const item = {
            id: ids[0],
            name: names[0],
            description: descriptions[0],
            sourceTypeId: sourceType,
            sourceTypeView: sourceTypeInfo.sourceTypeView,
          }
          EventBus.$emit('showEvents', {sourceInfo: item})
        }
        if (sourceTypeInfo.sourceTypeView === 0) {
          const sourceInfo = {
            id: ids[0],
            name: names[0],
            description: descriptions[0],
          }
          const data = {
            sourceTypeInfo,
            sourceInfo
          }
          EventBus.$emit('open-data-player', data);
        }
      }
    }
  }
  return {
    key: 'singleclick',
    action: (e) => handleShowPlayer(e)
  }
}
async update() {
  await this.updateFeaturesOnMap();
}
async updateFeaturesOnMap() {
  sharedData.mapLoading = true;
  const features = await this.getFeaturesForAll();
  if (features) {

```

					ПГУ1.090501.08.001 ПЗ	Лист
						110
Из	Лист	№ докум.	Подпись	Дата		

```

        const clusteredFeatures = clusterFeatures(features,
this.map);
        const source = new VectorSource({features:
clusteredFeatures.flat()});
        this.layer.setSource(source);
        sharedData.mapLoading = false;
    }
}
async getFeaturesForAll() {
    this.abortController.abort();
    this.abortController = new AbortController();
    const visibleSourceTypesId = this.sourceTypesInfo.filter(info
=> info.visibility).map(info => info.sourceType);
    const promises = visibleSourceTypesId.map((sourceTypeId) => {
        return this.getFeaturesForSourceType(sourceTypeId).catch(e
=> {
            if (e.name === 'CanceledError') {
                throw e;
            }
            return e;
        })
    });
    try {
        const results = await Promise.allSettled(promises);
        if (!Array.isArray(results)) return;
        if (results.some(r => r.status === 'rejected' &&
r.reason?.name === 'CanceledError')) {
            return undefined;
        }
        const featuresData = results.filter(r => r.status ===
'fulfilled');
        const filteredFeaturesData = featuresData?.flatMap(r =>
r.value)?.filter(Boolean);
        return filteredFeaturesData?.length ? filteredFeaturesData
: [];
    } catch {
        return undefined;
    }
}
async getFeaturesForSourceType(sourceTypeId) {
    let response;

    if (this.sourcesType === 'favorites') {
        response = await
TileService.getObjectsFavorite(sourceTypeId, this.extent,
this.getMaxPointsCountForSourceType(), this.abortController.signal);
    } else {
        response = await TileService.getObjects(sourceTypeId,
this.extent, this.getMaxPointsCountForSourceType(),
this.abortController.signal);
    }
    const features = response?.data?.features?.length ?
getFeaturesFromGeoJSON(response.data) : [];
    return features.map((f) => {
        const sourceTypeId = f.get('sourcetype');
        const inFavorites = f.get('infavorites');
    });
}

```

					ПГУ1.090501.08.001 ПЗ	Лист
						111
Из	Лист	№ докум.	Подпись	Дата		

```

        if (sourceTypeId !== undefined) {
            f.set('sourceType', sourceTypeId);
            f.unset('sourcetype');
        }
        if (inFavorites) {
            f.set('inFavorites', inFavorites || [false]);
            f.unset('infavorites');
        }
        return f;
    });
}
async setVisibilityForSourceType(sourceTypeId, visibility) {
    const sourceTypeInfo = this.sourceTypesInfo.find(info =>
info.sourceType === sourceTypeId);
    if (!sourceTypeInfo) return;

    sourceTypeInfo.visibility = visibility;

    await this.updateFeaturesOnMap();

    EventBus.$emit('onChangeExtent', {
        extent: this.extent,
        signal: this.abortController.signal
    });

    this.setUpdateHandlers();
}
getVisibilityForSourceType(sourceTypeId) {
    const sourceTypeInfo = this.sourceTypesInfo.find(info =>
info.sourceType === sourceTypeId);
    if (!sourceTypeInfo) return;

    return sourceTypeInfo.visibility;
}
async setGroupVisibility(sourceTypeView, visibility) {
    const sourceTypesInfoForView =
this.sourceTypesInfo.filter(info => info.sourceTypeView ===
sourceTypeView);
    sourceTypesInfoForView.forEach(info => info.visibility =
visibility);

    await this.updateFeaturesOnMap();

    EventBus.$emit('onChangeExtent', {
        extent: this.extent,
        signal: this.abortController.signal
    });

    this.setUpdateHandlers();
}
getLayer() {
    return this.layer;
}
getMaxPointsCountForSourceType() {
    const countVisibleSourceTypes =
this.sourceTypesInfo.filter(info => info.visibility).length;

```

					ПГУ1.090501.08.001 ПЗ	Лист
						112
Из	Лист	№ докум.	Подпись	Дата		

```

        return countVisibleSourceTypes ? Math.ceil(1000 /
this.sourceTypesInfo.length) : 1000;
    }
    setUpdateHandlers() {
        this.updateIntervalsId.forEach(intervalId =>
clearInterval(intervalId));
        this.updateIntervalsId.clear();

        this.sourceTypesInfo.forEach(info => {
            if (info.refresh > 0) {
                const intervalId = setInterval(async () => {
                    const featuresCount = await
this.getFeaturesForSourceType(info.sourceType).length;

                    const record =
this.lastFeaturesCount[info.sourceType];
                    if (record) {
                        if (featuresCount !== record) {
                            await this.updateFeaturesOnMap();
                        }
                    } else {
                        this.lastFeaturesCount[info.sourceType] =
featuresCount;
                    }
                }, info.refresh * 1000);
                this.updateIntervalsId.set(info.sourceType,
intervalId);
            }
        });
    }
}

```

Файл MovableLayerService.js

```

import Vue from "vue";
import {MovableLayerStyles} from
"@/services/MovableLayerService/MovableLayerStyles";
import VectorSource from "ol/source/Vector";
import VectorLayer from "ol/layer/Vector";
import {TileService} from "@/api/services/TileService";
import {debounce} from "@/utils/debounce";
import {getExtentForMapView, parseData} from
"@/services/MovableLayerService/MovableLayerUtils";
import {transform, transformExtent} from "ol/proj";
import {Feature} from "ol";
import {LineString, Point} from "ol/geom";
import {EventBus} from "@/utils/bus";
import {sharedData} from "@/utils/shared-data";
import {Style} from "ol/style";
import {getFeatureDataForPixel, toCoordinates} from
"@/utils/mapUtils";
import {errorHandlerNotify} from "@/utils/errorHandlerNotify";
export class MovableLayerService {
    constructor(sourceTypeInfo, sourceTypesInfo) {
        this.sourceTypeInfo = Vue.observable({
            ...sourceTypeInfo,
            visibility: sourceTypeInfo.defaultVisibility ?? true
        });
    }
}

```

					ПГУ1.090501.08.001 ПЗ	Лист
						113
Из	Лист	№ докум.	Подпись	Дата		

```

    });
    this.sourceTypeInfo = sourceTypeInfo;
    this.abortController = new AbortController();
    const {icon, icon_color} = this.sourceTypeInfo;
    this.styleService = new
MovableLayerStyles(this.sourceTypeInfo);
    this.source = new VectorSource();
    this.layer = new VectorLayer({source: this.source});
    this.lastSourcePositions = {};
    this.sources = undefined;
    this.boxData = undefined;
    this.pointFeatures = [];
    this.lineFeatures = [];
    this.trackBufferedPointFeatures = [];
    this.trackBufferedLineFeatures = [];
    this.map = undefined;
    this.extent = undefined;
    this.updateIntervalId = undefined;
    this.setUpdateInterval();
  }
  clearIntervals() {
    if (this.updateIntervalId) {
      clearInterval(this.updateIntervalId);
    }
  }
  setUpdateInterval() {
    if (this.updateIntervalId) {
      clearInterval(this.updateIntervalId);
    }
    if (!this.sourceTypeInfo.visibility) return;
    const refreshTime = this.sourceTypeInfo.refresh * 1000;
    if (refreshTime) {
      this.updateIntervalId = setInterval(async () => {
        await this.renderLayer();
      }, refreshTime)
    }
  }
  getEventsHandlers() {
    const handleExtentUpdate = debounce(async (event) => {
      const {map} = event;
      this.map = map;
      this.extent = getExtentForMapView(map);
      await this.renderLayer();
    }, 300);

    return [
      {
        key: 'moveend',
        action: (event) => {
          if (this.abortController) {
            this.abortController.abort();
          }
          this.abortController = new AbortController();
          handleExtentUpdate(event);
        }
      },
      {

```

					ПГУ1.090501.08.001 ПЗ	Лист
						114
Из	Лист	№ докум.	Подпись	Дата		


```

        key: 'singleclick',
        action: (event) => {
            event.preventDefault();
            event.stopPropagation();
            const {map} = event;
            const {feature} = getFeatureDataForPixel(map,
event.pixel);

            if (!feature) return;
            const sourceTypeInfo =
this.sourceTypesInfo.find(info => info.sourceType ===
feature.get('sourceType'));
            const sourceTypeView =
sourceTypeInfo.sourceTypeView;
            const isMovableFeature = sourceTypeView === 2;
            if (!feature || !isMovableFeature) return;
            const properties = feature.getProperties();
            const {sourceId, name, description} = properties;
            const sourceInfo = {
                id: sourceId,
                name,
                description,
            }
            let data = {
                sourceTypeInfo,
                sourceInfo
            }
            const isMain = feature.get('main');
            if (!isMain) {
                const eventInfo = {
                    eventId: feature.get('sourceId'),
                    eventName: feature.get('name'),
                    eventDescription:
feature.get('description'),
                    datetimeStart:
feature.get('datetimeStart'),
                    datetimeStop: feature.get('datetimeStop'),
                }
                data = {
                    ...data,
                    eventInfo,
                    playFragment: true
                }
            }
            EventBus.$emit('open-data-player', {...data,
playFragment: !!data.eventInfo});
        }
    ];
}
async renderLayer() {
    const response = await this.fetchDataForAllSources();
    if (!response) return;
    const {sourcesResponse = {}, objectsResponse = {}} = response;
    if (!sourcesResponse?.data?.length || !objectsResponse?.box ||
!objectsResponse?.features?.length) return;
    this.boxData = objectsResponse.box;
    const sources = sourcesResponse.data;

```

					ПГУ1.090501.08.001 ПЗ	Лист
						115
Из	Лист	№ докум.	Подпись	Дата		

```

        this.sources = sources;
        const objects = objectsResponse.features;
        this.lineFeatures = [];
        this.pointFeatures = [];
        await this.getLastTracksFeaturesForAllSources(sources,
objects);
        this.source.clear();
        this.source.addFeatures([...this.pointFeatures,
...this.lineFeatures, ...this.trackBufferedLineFeatures,
...this.trackBufferedPointFeatures]);
    }
    async getLastTracksFeaturesForAllSources(sources, objects) {
        const parsedData = await parseData(sources, objects);
        this.pointFeatures = [];
        parsedData.forEach(source => {
            const lastPoint = source.tracks?.at(-1)?.points?.at(-1);
            if (lastPoint) {
                const coordinates = transform(lastPoint.coordinates,
'EPSG:4326', 'EPSG:3857');
                this.lastSourcePositions[source.id] = coordinates;
                const pointFeature = new Feature({
                    geometry: new Point(coordinates),
                    ...lastPoint,
                    sourceId: source.id,
                });
                const prevPoint = source.tracks?.at(-2)?.points?.at(-
2);
                pointFeature.setStyle(this.sourceTypeInfo.visibility ?
() => this.styleService.point(lastPoint, prevPoint) : new Style(null));
                this.pointFeatures.push(pointFeature);
            }
        })
    }
    async showTrackOnMap(trackId, trackName) {
        if (!this.sources) return;
        this.removeTimeShowedTrack();
        let firstTrackPoint = undefined;
        try {
            sharedData.mapLoading = true;
            const {data: {features}} = await
TileService.getGeoDataForEvent(this.sourceTypeInfo.sourceType, trackId);
            const parsedData = parseData(this.sources, features,
trackName);
            const sourceId = parsedData[0].id;
            const sourceName = parsedData[0].name;
            const sourceDescription = parsedData[0].description;
            const track = parsedData[0]?.tracks[0];
            if (track?.points) {
                track.points = track.points.map(p => ({...p, main:
false}));
            }
            if (!track) return;
            const trackCoordinates = track.points.map(point => {
                return transform(point.coordinates, 'EPSG:4326',
'EPSG:3857');
            });
            trackCoordinates.forEach((point, index) => {

```

					ПГУ1.090501.08.001 ПЗ	Лист
						116
Из	Лист	№ докум.	Подпись	Дата		

```

        let featureProperties = {
            geometry: new Point(point),
            timeShowed: true,
            sourceId: sourceId,
            ...track.points[index],
        }
        if (index < trackCoordinates.length - 1) {
            featureProperties = {
                ...featureProperties,
                datetimeStart: track.points[index].dateTime,
                datetimeStop: track.points[index + 1].dateTime
            }
        }
        const pointFeature = new Feature(featureProperties);
        if (index === 0) firstTrackPoint = pointFeature;

        const prevPoint = track.points[index - 1];
        pointFeature.setStyle(() =>
this.styleService.point(track.points[index], prevPoint));
        this.trackBufferedPointFeatures.push(pointFeature);
        if (index < trackCoordinates.length - 1) {
            const lineFeature = new Feature({
                geometry: new LineString([point,
trackCoordinates[index + 1]]),
                timeShowed: true,
                trackId: track.id,
                sourceType: this.sourceTypeInfo.sourceType,
                name: sourceName,
                description: sourceDescription,
                datetimeStart: track.points[index].dateTime,
                datetimeStop: track.points[index +
1].dateTime,

                sourceId: sourceId,
                trackName

            });
            lineFeature.setStyle(this.styleService.line);
            this.trackBufferedLineFeatures.push(lineFeature);
        }
    });

this.source.addFeatures([...this.trackBufferedLineFeatures]);

this.source.addFeatures([...this.trackBufferedPointFeatures]);
    if (trackCoordinates.length < 2) return;
    sharedData.showTrackState = {
        active: true,
        removeFunc: this.removeTimeShowedTrack.bind(this)
    }
    toCoordinates(this.map,
firstTrackPoint.getGeometry().getCoordinates(), true);
    } catch (e) {
        errorHandlerNotify(e, "Просмотр маршрута на карте");
    } finally {
        sharedData.mapLoading = false;
    }
}
removeTimeShowedTrack() {

```

					ПГУ1.090501.08.001 ПЗ	Лист
						117
Из	Лист	№ докум.	Подпись	Дата		

```

const removeFromSource = (source, features) => {
  for (let i = features.length - 1; i >= 0; i--) {
    if (features[i].get('timeShowed')) {
      source.removeFeature(features[i]);
      features.splice(i, 1);
    }
  }
};
removeFromSource(this.source,
this.trackBufferedPointFeatures);
removeFromSource(this.source, this.trackBufferedLineFeatures);
sharedData.showTrackState = {
  active: false,
  removeFunc: undefined
};
this.trackBufferedPointFeatures = [];
this.trackBufferedLineFeatures = [];
this.source.changed();
}
async fetchDataForAllSources() {
  if (!this.extent || !this.abortController) return;
  try {
    const sourceTypeId = this.sourceTypeInfo.sourceType;
    const [sources, objects] = await Promise.all([
      TileService.getSourcesForSourceTypeId(sourceTypeId),
      TileService.getGeoDataForSourceTypeId(sourceTypeId,
this.extent, this.abortController.signal),
    ]);
    return {
      sourcesResponse: sources?.data ?? [],
      objectsResponse: objects?.data ?? []
    };
  } catch (error) {
    if (error.name === 'CanceledError') return;
    console.error('При получении геоданных для подвижного слоя
возникла ошибка', error);
    throw new Error('При получении геоданных для подвижного
слоя возникла ошибка');
  }
}
centerForSource(sourceId) {
  const sourceCoordinates = this.lastSourcePositions[sourceId];
  if (!sourceCoordinates) return;
  this.map.getView().animate({
    center: sourceCoordinates,
    zoom: 17,
    duration: 1000
  })
}
center(map) {
  if (!map || !this.boxData) return;
  const extent = [this.boxData.xmin, this.boxData.ymin,
this.boxData.xmax, this.boxData.ymax];
  const transformedExtent = transformExtent(extent, 'EPSG:4326',
map.getView().getProjection());
  map.getView().fit(transformedExtent, {
    size: map.getSize(),

```

					ПГУ1.090501.08.001 ПЗ	Лист
						118
Из	Лист	№ докум.	Подпись	Дата		

```

        padding: [50, 50, 50, 50],
        duration: 500,
    });
}
getLayer() {
    return this.layer;
}
setVisibility(bool) {
    this.sourceTypeInfo.visibility = bool;
    this.layer.setVisible(bool);
    const visibleSourceTypesId = this.sourceTypesInfo.filter(info
=> info.visibility).map(info => info.sourceType);
    const extent = getExtentForMapView(this.map);
    EventBus.$emit('onChangeExtent', {visibleSourceTypesId,
extent, signal: this.abortController.signal});
    this.setUpdateInterval();
}
getVisibility() {
    return this.sourceTypeInfo.visibility;
}
}

```

Файл clusterFeatures.js

```

import {Feature} from "ol";
import {Point} from "ol/geom";
function clusterFeatures(features, map) {
    const clusters = [];
    const clusterRadiusSquared = 35 * 35; // Сравниваем квадрат
радиуса (для избегания sqrt)
    const processed = new Set(); // Множество для отслеживания уже
обработанных точек
    const view = map.getView();
    // Массив для хранения пиксельных координат
    const pixelCoords = [];
    for (let i = 0; i < features.length; i++) {
        const geometry = features[i].getGeometry();
        if (!geometry) continue;
        const coords = geometry.getCoordinates();
        if (Array.isArray(coords)) {
            pixelCoords.push(map.getPixelFromCoordinate(coords));
        } else {
            pixelCoords.push(null); // Для тех, у кого нет координат,
сохраняем null
        }
    }
    // Функция для вычисления квадратного расстояния между пиксельными
координатами
    function getPixelDistanceSquared(pixel1, pixel2) {
        if (!pixel1 || !pixel2) return Infinity; // Проверка на null
        const dx = pixel1[0] - pixel2[0];
        const dy = pixel1[1] - pixel2[1];
        return dx * dx + dy * dy;
    }
    // Основной цикл по объектам
    for (let i = 0; i < features.length; i++) {

```

					ПГУ1.090501.08.001 ПЗ	Лист
						119
Из	Лист	№ докум.	Подпись	Дата		

```

        if (processed.has(features[i])) continue; // Пропускаем уже
        обработанные объекты
        const cluster = [];
        const centerPixel = pixelCoords[i];
        if (!centerPixel) continue; // Если пиксельные координаты
        отсутствуют, пропускаем
        // Поиск соседей для текущего объекта
        for (let j = 0; j < features.length; j++) {
            if (processed.has(features[j])) continue;
            const targetPixel = pixelCoords[j];
            if (getPixelDistanceSquared(centerPixel, targetPixel) <=
            clusterRadiusSquared) {
                cluster.push(features[j]);
                processed.add(features[j]);
            }
        }
        // Если кластер содержит несколько объектов, вычисляем его
        центр
        if (cluster.length > 1) {
            const coordinates = cluster.map(f =>
            f.getGeometry().getCoordinates());
            const clusterCenter = getClusterCenter(coordinates);
            const clusterFeature = new Feature({
                geometry: new Point(clusterCenter),
                features: cluster,
                size: cluster.length,
            });
            clusters.push(clusterFeature);
        } else if (cluster.length === 1) {
            clusters.push(cluster[0]);
        }
    }
    return clusters;
}
function getClusterCenter(coordinates) {
    const sum = coordinates.reduce((acc, coord) => {
        acc[0] += coord[0];
        acc[1] += coord[1];
        return acc;
    }, [0, 0]);
    return [sum[0] / coordinates.length, sum[1] / coordinates.length];
}
export default clusterFeatures;

```

Файл MapContainer.js

```

<template>
  <div style="height: 100%; width: 100%; position: relative;">
    <div class="map-renderer" ref="mapContainer">
      <div id="popover"></div>
      <LayerSwitcher :templates="templates"/>
    </div>
    <v-progress-linear color="primary-color" indeterminate v-
    if="sharedData.mapLoading"
                        style="position: absolute; bottom: 0;"
    />
  </div>

```

					ПГУ1.090501.08.001 ПЗ	Лист
						120
Из	Лист	№ докум.	Подпись	Дата		

```

</template>
<script>
import 'ol/ol.css';
import Map from 'ol/Map';
import View from 'ol/View';
import {sharedData} from "@/utils/shared-data";
import {defaults, MouseWheelZoom} from "ol/interaction";
import LayerSwitcher from
"@/features/LayerSwitcher/LayerSwitcher.vue";
import {EventBus} from "@/utils/bus";
import {fromLonLat, get, Projection} from "ol/proj";
import {FeaturesPopupService} from
"@/services/FeaturesPopupService/FeaturesPopupService";
import {ContextMenuService} from
"@/services/ContextMenuService/ContextMenuService";
import {FullScreen, MousePosition, ScaleLine, ZoomSlider} from
"ol/control";
import {defaults as defaultControls} from 'ol/control/defaults';
import {format} from "ol/coordinate";
export default {
  name: 'Map',
  components: {LayerSwitcher},
  computed: {
    sharedData() {
      return sharedData
    }
  },
  props: ['layers', 'eventsHandlers', 'loading', 'templates',
'sourceTypesInfo', 'contextMenuOptions', 'resultLayer'],
  data() {
    return {
      map: null,
      resizeObserver: null,
      popupManager: null,
      contextMenuService: undefined
    };
  },
  watch: {
    resultLayer(newLayer) {
      if (newLayer) {
        this.map.addLayer(newLayer);
      }
    }
  },
  mounted() {
    this.initializeMap();
    this.popupManager = new FeaturesPopupService(this.map,
this.sourceTypesInfo);
    this.setupResizeObserver();
    this.layers.forEach(l => this.map.addLayer(l));
    this.setupEventsHandlers();
    this.contextMenuService = new ContextMenuService(this.map,
this.contextMenuOptions, this.sourceTypesInfo);
    EventBus.$on('toCoordinates', async ({coordinates}) => {
      this.map.getView().animate({
        center: fromLonLat(coordinates),
        zoom: 17,

```

					ПГУ1.090501.08.001 ПЗ	Лист
						121
Из	Лист	№ докум.	Подпись	Дата		

```

        duration: 1000
    })
    });
},
beforeDestroy() {
    EventBus.$off('toCoordinates');
    if (this.resizeObserver) {
        this.resizeObserver.disconnect();
    }
    this.contextMenuService.removeHandlers();
},
methods: {
    mousePositionControl() {
        return new MousePosition({
            coordinateFormat: (coordinate) => {
                return format(coordinate, '{y}', '{x}', 6);
            },
            projection: 'EPSG:4326',
            className: 'custom-mouse-position',
        });
    },
    initializeMap() {
        const mouseWheelZoom = new MouseWheelZoom({
            duration: 250,
            constrainResolution: true,
            maxDelta: 2
        });
        this.map = new Map({
            controls: defaultControls().extend([
                new FullScreen(),
                new ZoomSlider(),
                new ScaleLine(),
                this.mousePositionControl(),
            ]),
            target: this.$refs.mapContainer,
            interactions: defaults().extend([mouseWheelZoom]),
            layers: [],
            view: new View({
                center: fromLonLat([30.153151, 59.958610]),
                zoom: 10,
                constrainOnlyCenter: false,
                extent: [-20026376.39, -20048966.10, 20026376.39,
20048966.10],
                projection: new Projection({
                    code: 'EPSG:3857',
                    units: 'm',
                    global: false,
                    extent: get('EPSG:3857').getExtent()
                })
            })
        });
        this.$emit('onMapInit', this.map);
    },
    setupResizeObserver() {
        this.resizeObserver = new ResizeObserver(() => {
            if (this.map) {
                this.map.updateSize();
            }
        });
    }
}

```

					ПГУ1.090501.08.001 ПЗ	Лист
						122
Из	Лист	№ докум.	Подпись	Дата		


```

        }
    });
    this.resizeObserver.observe(this.$refs.mapContainer);
  },
  setupEventHandlers() {
    this.eventsHandlers?.forEach(handler => this.map.on(handler.key,
handler.action));
  }
},
};
</script>
<style>
.map-renderer {
  width: 100%;
  height: 100%;
}
.popover {
  width: 250px !important;
}
.popover-body {
  padding: 5px 10px !important;
}
.popover-header {
  font-size: 14px !important;
}
.custom-mouse-position {
  position: absolute;
  bottom: 8px;
  right: 8px;
  padding: 2px;

  background-color: var(--ol-partial-background-color);
  border-radius: 4px;
  width: 140px;
  text-align: center;
  font-family: Arial, sans-serif;
  font-size: 12px;
}
</style>

```

					ПГУ1.090501.08.001 ПЗ	Лист
						123
Из	Лист	№ докум.	Подпись	Дата		

Приложение В

(Обязательное)

Клиентская часть геоинформационной платформы сбора данных

Презентация

					ПГУ1.090501.08.001 ПЗ	Лист
						124
Из	Лист	№ докум.	Подпись	Дата		

Клиентская часть геоинформационной платформы сбора данных с интеграцией медиаконтента

Выполнил: студент гр. 20ВОЭ1
Голышев А.А.

Научный руководитель: д.т.н., профессор каф. ВТ
Зинкин С.А.

Цель:

Проектирование и разработка клиентской части геоинформационной платформы для мониторинга на картографической подложке статичных и подвижных физических устройств с возможностью просмотра мультимедийных данных.

Задачи:

- Реализация отображения картографических подложек;
- Производительное отображение статичных объектов с поддержкой кластеризации;
- Визуализация подвижных объектов с отображением маршрутов движения;
- Интеграция медиаплееров для просмотра фото- и видеоконтента;
- Обеспечение фильтрации объектов по выделенной области;
- Внедрение механизма аутентификации и безопасной авторизации;
- Контейнеризация и подготовка приложения к внедрению с использованием Docker и Nginx.

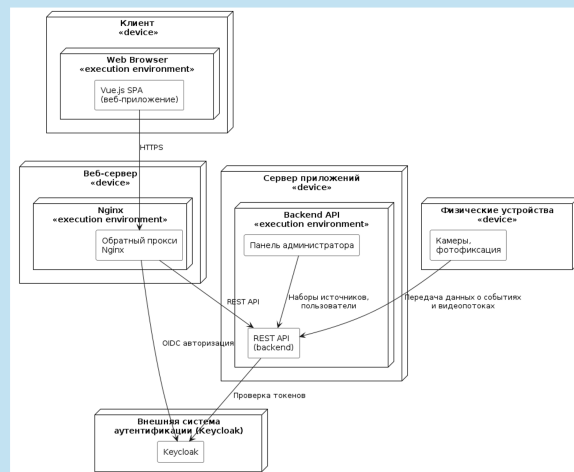
Актуальность разработки

- Рост объема геопространственных данных от распределённых физических устройств требует новых подходов к их обработке и визуализации;
- Существующие ГИС-платформы на российском рынке обладают ограниченной поддержкой видеоконтента и подвижных объектов;
- Необходима интеграция мультимедийных данных (фото, видео) с картографической информацией для повышения наглядности и оперативности анализа;
- Актуален запрос на автономные решения, способные функционировать в локальной инфраструктуре без зависимости от внешних и зарубежных сервисов.

					ПГУ1.090501.08.001 ПЗ	Лист
						125
Из	Лист	№ докум.	Подпись	Дата		

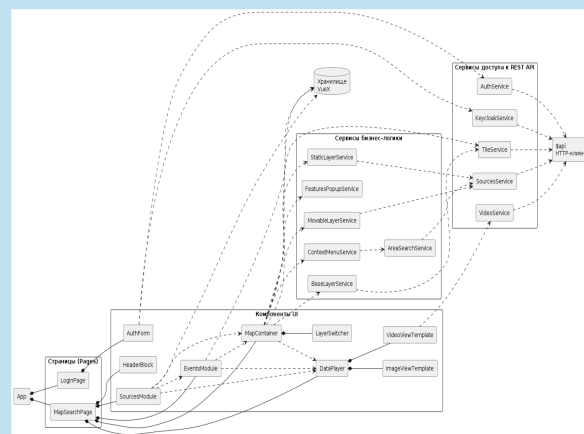
Общая архитектура системы

- **Модульная структура**
Каждый компонент изолирован, легко масштабируется и заменяется;
- **Контейнеризация (Docker)**
Все части системы развернуты в виде контейнеров: frontend, backend, Keycloak;
- **Frontend (Vue + Nginx)**
Реализует интерфейс пользователя и обрабатывает сетевые запросы через Nginx;
- **Backend (REST API)**
Обрабатывает данные от устройств и отвечает за бизнес-логику приложения;
- **Аутентификация через Keycloak**
Безопасная авторизация пользователей с использованием JWT-токенов.



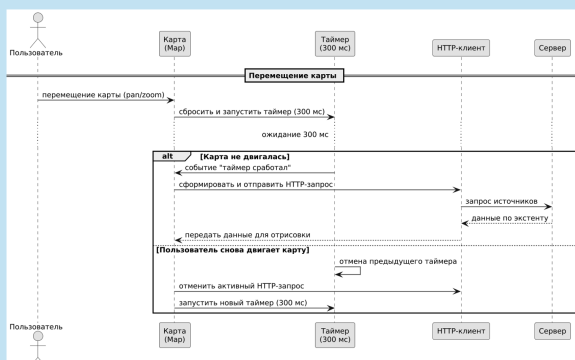
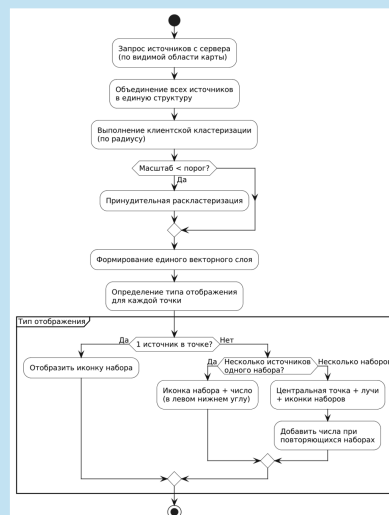
Технологический стек

- **Vue + Vuetify (SPA)**
Быстрая, кроссплатформенная клиентская часть с адаптивным интерфейсом;
- **OpenLayers**
Гибкая карта с высокой производительностью, работает без Интернета;
- **Keycloak + keycloak-js**
Безопасная аутентификация, поддержка OpenID Connect;
- **Axios**
Унифицированные HTTP-запросы, обработка ошибок, автообновление токенов;
- **Video.js**
Воспроизведение видео в прямом эфире и из архива;
- **Docker**
Контейнеризация всех компонентов для автономной и масштабируемой архитектуры.

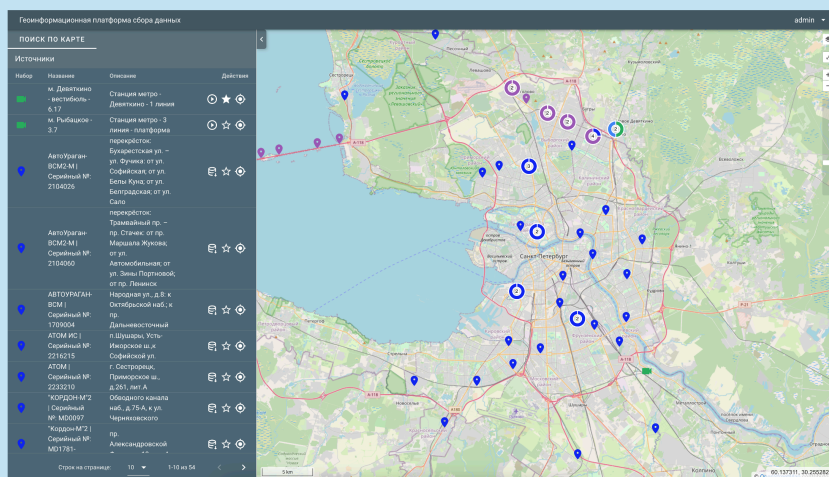


Отображение статичных объектов

- Оптимизация сетевого взаимодействия;
- Оптимизация визуализации объектов при их высокой плотности.

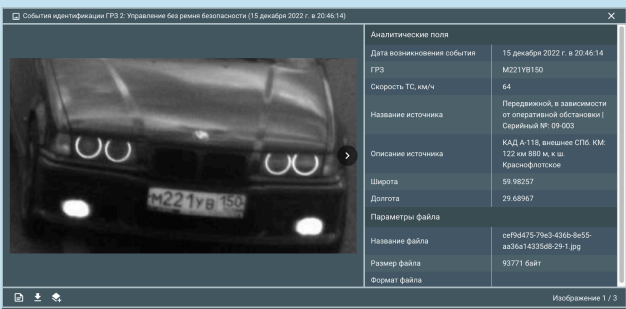


Отображение статичных объектов



Фотоплеер

- **Просмотр фотографий, привязанных к событиям**
Удобный слайдер позволяет листать фото прямо в интерфейсе карты;
- **Скачивание изображений на устройство**
Пользователь может сохранить интересующее фото одним нажатием кнопки;
- **Просмотр метаданных события**
Есть доступ к дополнительной информации: дата, источник, описание и др.



Видеоплеер

- **Поддержка прямой трансляции и архивного просмотра**
Воспроизведение видео с камер в реальном времени и за выбранный период;
- **Интерактивный индикатор прямого эфира**
Подсказка статуса эфира с возможностью синхронизации одним кликом;
- **Отображение реального календарного времени**
Позволяет точно соотнести происходящее на видео с конкретным моментом времени;
- **Автоматическое обновление токенов доступа**
Обеспечивает непрерывный просмотр без необходимости вручную перезапускать видео;
- **Восстановление при ошибках видео**
В случае повреждённого сегмента видео выполняется автоматическая перемотка и повтор загрузки;
- **Интеграция в общий интерфейс системы**
Видеоплеер встроен в рабочее окно оператора и доступен при выборе соответствующего источника.



Развертывание и инфраструктура

- **Контейнеризация через Docker**
Приложение развёртывается в контейнере, не зависит от ОС и легко переносится между серверами;
- **Сборка в два этапа**
Сначала — создание приложения (Node.js), затем — упаковка с веб-сервером (Nginx);
- **Nginx как прокси и сервер интерфейса**
Отвечает за HTTPS, маршрутизацию запросов и подачу статических файлов.
- **Безопасность данных**
Реализована поддержка SSL-сертификатов и современных криптографических протоколов (TLS 1.2/1.3);
- **Гибкая настройка через переменные окружения**
Один образ можно использовать в разных средах — адреса API подставляются при запуске, без пересборки;
- **Готово для реального развёртывания**
Настроен скрипт запуска, заменяющий конфигурации под окружение и автоматически запускающий сервер.

Заключение

Основные итоги

- Разработана клиентская часть геоинформационной платформы с интеграцией фото- и видеоконтента;
- Реализована авторизация через Keycloak и JWT с безопасной работой в изолированной инфраструктуре;
- Визуализированы статичные и подвижные объекты с маршрутизацией, фильтрацией и кластеризацией;
- Встроены медиаплееры для фото и видео с обработкой ошибок и прямыми трансляциями;
- Настроено развёртывание через Docker и Nginx с поддержкой HTTPS и гибкой конфигурацией.

Практическая значимость

- Применимо в системах мониторинга, логистики, безопасности;
- Готово к использованию в закрытых ведомственных сетях;
- Поддержка расширения функциональности и масштабируемость.

					ПГУ1.090501.08.001 ПЗ	Лист
						130
Из	Лист	№ докум.	Подпись	Дата		