Оглавление

ВВЕДЕНИЕ

За последние сто лет человечество достигло значительных высот в освоении космического пространства. Особенно заметны эти достижения на территории Российской Федерации, где общественное сознание в первую очередь ассоциирует космическую программу с яркими вехами — полётом Юрия Гагарина и первым выходом человека в открытый космос. Однако за этими событиями стоит многогранная и сложная система: десятки институтов, сотни проектов, тысячи специалистов, вклад которых остался в тени истории. Сегодня отсутствует единый наглядный ресурс, позволяющий структурировано и удобно изучать эволюцию ракетно-космической отрасли.

Для восполнения этого пробела разработана подсистема «Навигатор», входящая в состав многофункциональной информационной системы, посвящённой становлению и развитию отечественной космонавтики. Её основная задача — предоставить пользователю интуитивный инструмент визуального анализа и поиска информации о ключевых объектах и событиях ракетно-космической отрасли, с возможностью построения маршрутов между ними. Система ориентирована как на исследователей и специалистов, так и на широкую аудиторию, интересующуюся историей космоса.

Созданная платформа реализована в виде геоинформационной системы (ГИС), способной отображать широкий спектр объектов, связанных с космической индустрией, и формировать маршруты на основе пользовательских предпочтений. Архитектура решения включает клиентскую часть, обеспечивающую интерактивное взаимодействие, и серверную, отвечающую за обработку, структурирование и кластеризацию данных с целью оптимизации визуализации.

Одним из ключевых вызовов при реализации подсистемы стало обеспечение высокой производительности при отображении большого объёма геопространственной информации. Масштабный объём данных требует значительных вычислительных ресурсов, что при прямой обработке на стороне

клиента приводит к снижению отзывчивости интерфейса и ухудшению пользовательского опыта.

Для преодоления этой проблемы был разработан специализированный алгоритм, минимизирующий вычислительную нагрузку на клиентской стороне за счёт предварительной обработки и интеллектуальной кластеризации данных на сервере. Дополнительным преимуществом предложенного подхода является его универсальность: система способна работать с разнообразными типами информации, обеспечивая масштабируемость и расширяемость платформы.

.

1. ПРЕДПРОЕКТНОЕ ОБСЛЕДОВАНИЕ

В данной главе проводится анализ исходных положений и требований к разрабатываемой подсистеме «Навигатор», входящей в состав многофункциональной информационной системы по истории становления и развития ракетно-космической отрасли. В рамках обследования определяются цели проектирования, анализируется предметная область, устанавливаются критерии качества, проводится сравнительный анализ аналогичных решений и формируется перечень ключевых задач, подлежащих реализации в процессе разработки.

1.1 Постановка задачи проектирования

Задача проектирования подсистемы «Навигатор» заключается в создании высокопроизводительного и масштабируемого программного модуля, обеспечивающего интерактивное отображение объектов ракетно-космической отрасли на карте, а также построение маршрутов между ними на основе пользовательских запросов. Подсистема должна интегрироваться в состав многофункциональной информационной системы, ориентированной на структурирование и визуализацию информации о развитии отечественной космонавтики.

Проектируемая система должна обеспечивать быструю и удобную работу с большим объёмом геопространственных данных, предоставляя пользователю возможность фильтрации объектов по различным параметрам, просмотра атрибутивной информации и построения маршрутов. При этом ключевым требованием является минимизация нагрузки на клиентскую часть при одновременном обеспечении высокой точности отображения и стабильной производительности.

В рамках проектирования необходимо решить следующие технические задачи: организация эффективной архитектуры взаимодействия между клиентом

и сервером; выбор технологий для хранения, обработки и визуализации данных; реализация алгоритмов кластеризации и маршрутизации; а также обеспечение возможности гибкой фильтрации и масштабирования системы. Особое внимание должно быть уделено предварительной серверной обработке данных с применением кластеризации (например, DBSCAN), пространственной индексации (Geohash), интеллектуального кэширования (Redis) и построения маршрутов с использованием API картографического сервиса.

Таким образом, цель проектирования заключается в разработке технологически устойчивой подсистемы, способной обрабатывать массивы данных в режиме реального времени, обеспечивая при этом адаптивную визуализацию и навигацию по объектам в рамках ГИС-интерфейса.

1.2 Описание предметной области

1.2.1 Естественно-языковая модель предметной области

Предметная область данной работы охватывает объекты и инфраструктуру, связанные с ракетно-космической отраслью. Это могут быть различные институты, исследовательские центры, космодромы, музеи и памятные места, которые сыграли важную роль в истории отечественной космонавтики. Все эти объекты не просто существуют в виде списка — они имеют чёткие координаты, находятся в разных регионах страны и обладают определёнными характеристиками: тип, назначение, доступность, историческая значимость и так далее.

Для пользователя подобные объекты могут быть интересны с разных точек зрения: кто-то изучает историю отрасли, кто-то планирует тематические маршруты, а кто-то просто хочет увидеть, где находятся важнейшие точки развития космонавтики. Актуальной задачей становится создание системы, которая могла бы не только хранить информацию об этих объектах, но и

отображать их на карте, объединять в маршруты и позволять фильтровать по интересующим признакам.

1.2.2 Выделенные сущности и связи

Для создания модели предметной области необходимо выделить следующие ключевые сущности:

- **Объект ракетно-космической отрасли.**

Данная сущность является основной в системе, она представляет собой конкретное место, которое связано с деятельностью отрасли. Это могут быть музеи, космодромы, лаборатории, институты и другие места, связанные с РКО. Каждый объект, обладает уникальными свойствами: координаты, описательная информация, которая включает в себя наименование, тип, краткое описание и фотографии с данного места.

- **Тип объекта.**

Для классификации и упрощения фильтрации, объекты обладаем своими подтипами. Тип отражает принадлежность к какой-то категории, например: «космодром», «исторический объект», «музей» и другие. Данная сущность, является справочником, который позволяет ссылаться на конкретные объекты. Это позволяет упростить добавление новые категорий, без изменения в основной модели, также обеспечивая расширяемость системы.

- **Маршрут.**

Маршрут представляет собой упорядоченный набор точек, требуемые для построения навигации между объектами. Система поддерживает как маршруты, предоставляемые изначально системой, так и маршруты, созданные вручную пользователем. Маршрут содержит общую информацию, включая его название и описание промежуточных точек, для возможности изменения последовательности посещения.

- **Точка маршрута.**

Точка маршрута является вспомогательной сущностью, отражающей конкретный объект, входящий в состав маршрута, а также его позицию в последовательности движения. Несмотря на то, что каждая точка маршрута ссылается на конкретный объект отрасли, она не является самим объектом. Это структурный элемент маршрута, предназначенный для хранения порядка следования и дополнительного контекста (например, пояснений или временных ограничений). Таким образом, один и тот же объект может использоваться в составе нескольких различных маршрутов, занимая в них разные позиции. Это позволяет гибко формировать маршруты под различные сценарии использования, не дублируя информацию об объектах.

Взаимодействие между сущностями организовано через систему логических связей:

- Каждый объект принадлежит одному из типов, что обеспечивает классификацию и возможность тематической фильтрации.
- Один маршрут состоит из множества точек, каждая из которых указывает на конкретный объект.
- Один пользователь может быть автором нескольких маршрутов.
- Каждая точка маршрута сохраняет ссылку на соответствующий объект и своё положение в маршруте.

Продемонстрированная структура, позволяет обеспечить системе модульность, масштабируемость и ее расширяемость. Формирует представление о структуре представляемой информации и логике ее взаимодействия пользователя с интерфейсом навигатора.

1.2.3 Существующие технологии обработки информации и принятия управленческих решений

В условиях быстрого развития цифровых технологий, значительное внимание уделяется созданию информационных систем, способных обрабатывать большие объемы пространственных данных и оказывать помощь в

принятии управленческих решений. Особую роль в этом процессе играют геоинформационные технологии, методы интеллектуальной агрегации информации и алгоритмы маршрутизации, а также современные решения в области высокопроизводительных хранилищ данных.

Современные ГИС-системы выступают универсальным инструментом, обеспечивающим сбор, хранение, анализ и визуализацию пространственно-распределённых данных. Благодаря использованию расширений для реляционных СУБД, таких как PostGIS, возможно выполнение сложных пространственных запросов и операций над геообъектами непосредственно на уровне базы данных. Эти технологии позволяют реализовывать гибкие фильтры по координатам, учитывать особенности рельефа и логики размещения объектов, что особенно важно для навигационных задач, связанных с территориально распределёнными элементами инфраструктуры ракетно-космической отрасли.

Наряду с этим, одной из тенденций в разработке ГИС-систем является перенос части вычислительной нагрузки на сервер. Такой подход обеспечивает более стабильную работу клиентской части, особенно в условиях отображения и обработки большого количества объектов. Серверная логика позволяет реализовать продвинутые механизмы кластеризации и агрегирования данных, что особенно актуально при взаимодействии с высоконагруженными интерфейсами. Алгоритмы вроде DBSCAN находят широкое применение в этих сценариях, поскольку позволяют группировать объекты на основе их пространственной плотности без необходимости заранее задавать количество кластеров.

Отдельного внимания заслуживает использование in-memory технологий для хранения и ускоренного доступа к данным. Одним из наиболее эффективных решений в этой области является Redis, поддерживающий специализированные структуры данных для работы с географическими координатами. Это позволяет в реальном времени производить выборку объектов, находящихся в пределах заданного радиуса или области карты, с минимальными задержками. Кроме того, использование механизма временного хранения (TTL) способствует

оптимизации работы системы за счёт автоматического удаления устаревших записей, например, временных маршрутов.

Важнейшим элементом логики принятия решений в навигационных системах является построение оптимального маршрута. Алгоритмы, реализующие эту задачу, позволяют найти кратчайший путь между объектами с учетом различных ограничений. В данной работе используется алгоритм Дейкстры, который, благодаря своей простоте и эффективности, остаётся актуальным инструментом при работе с дорожными графами, включая те, что предоставляются сторонними API (например, Яндекс.Карты). Внедрение таких алгоритмов позволяет учитывать не только геометрию пути, но и параметры, влияющие на предпочтительность маршрута, такие как средняя скорость передвижения, тип транспортного средства и доступность точек.

Таким образом, совокупность вышеописанных технологий формирует комплексную информационную платформу, позволяющую не только отображать данные, но и предоставлять пользователю инструменты для анализа и принятия решений. Возможность фильтрации объектов, динамическое построение маршрутов, визуализация плотности размещения и агрегация информации в зависимости от масштаба — все эти элементы делают систему не просто картографическим интерфейсом, а полноценным средством поддержки управленческих решений в контексте территориально распределённой инфраструктуры ракетно-космической отрасли.

1.2.4 Функции, которые будут автоматизированы

В процессе разработки системы основное внимание уделяется максимально эффективному упрощению работы с большими объемами данных, относящихся к объектам ракетно-космической отрасли. Основная цель — сделать так, чтобы пользователю не приходилось вручную искать, фильтровать

или как-то отдельно управлять информацией. Ниже я описываю, какие именно функции удалось автоматизировать, и зачем это вообще нужно.

Автоматическая отрисовка объектов на карте

Когда пользователь открывает карту, система сама подгружает нужные объекты в зависимости от того, какой участок карты в данный момент виден. Никаких кнопок «обновить» или ручной загрузки не нужно — всё происходит «на лету» при перемещении или масштабировании карты.

Кластеризация точек

Если объектов становится слишком много (например, пользователь отдаляет карту и видит сразу большую территорию), система автоматически объединяет близко расположенные точки в кластеры. Это сильно разгружает интерфейс и помогает визуально понять, где на карте плотность объектов выше.

Геопоиск по области

При каждом изменении области карты система сама ищет только те объекты, которые попадают в эту зону. Не нужно загружать все данные сразу — это экономит ресурсы и ускоряет работу.

Построение маршрутов

Пользователь может выбрать несколько интересных точек, и система автоматически построит маршрут между ними. Всё работает через API карт, поэтому учитываются реальные дороги и расстояния, как в навигаторах.

Фильтрация объектов по типу

Если нужно посмотреть, например, только музеи или только стартовые площадки — достаточно включить нужный фильтр. Система покажет только подходящие объекты и сразу пересчитает видимую часть карты.

Сохранение маршрутов

После того как пользователь собрал свой маршрут, его можно сохранить. Всё — последовательность точек, названия, дополнительная информация — сохраняется автоматически, и маршрут можно будет открыть позже или отправить кому-то ещё.

Умная подгрузка данных

Чем больше объектов, тем сложнее всё быстро отрисовать. Поэтому система сама решает, сколько данных нужно подгрузить, в зависимости от масштаба и плотности объектов на карте. Это помогает сделать интерфейс плавным и не перегружать браузер.

1.3 Выбор и обоснование критериев качества

При разработке информационных систем важно не только реализовать функциональность, но и понимать, по каким параметрам можно оценить, насколько система получилась качественной. В данном случае это особенно актуально, так как система должна работать с большим количеством геоданных, оставаться быстрой, отзывчивой и удобной для пользователя, несмотря на потенциально высокую нагрузку.

Было выделино несколько ключевых критериев, которые позволяют объективно оценить качество разрабатываемой системы:

1. **Производительность (отклик системы)** — это время, за которое пользователь получает результат после запроса (например, отображения объектов или построения маршрута). Важно, чтобы даже при большом количестве объектов интерфейс оставался отзывчивым.
2. **Масштабируемость** — способность системы эффективно работать как с сотней объектов, так и с тысячами. Особенно это критично для карты, где количество точек может быть очень большим.
3. **Точность кластеризации** — насколько адекватно алгоритм объединяет близкорасположенные объекты. Плохая кластеризация может ввести пользователя в заблуждение или перегрузить карту.
4. **Корректность маршрутов** — маршруты должны строиться логично и корректно с учётом реальных расстояний, без лишних пересечений и "прыжков" между точками.

5. **Понятность интерфейса** — даже технически сложная система должна оставаться простой в использовании. Это субъективный критерий, но его можно оценить через пользовательское тестирование.
6. **Скорость геопоиска** — время, за которое система возвращает объекты из заданной области карты. Этот критерий напрямую связан с Redis и его гео-командами.

1.4 Анализ аналогов и прототипов

Прежде чем разрабатывать систему, было проанализировано, какие уже существуют решения, связанные с отображением объектов на карте и построением маршрутов. На первый взгляд, кажется, что всё уже есть — Yandex.Карты, Google Maps, 2ГИС и множество других сервисов умеют показывать объекты, строить маршруты и даже предоставляют API для работы с геоданными. Однако при более внимательном рассмотрении выясняется, что ни одно из этих решений не подходит под задачи, связанные с ракетно-космической отраслью.

Во-первых, такие системы ориентированы на массового пользователя. Они хорошо работают для поиска кафе, больниц или офисов, но не умеют обрабатывать отраслевые объекты — такие как стартовые площадки, исследовательские институты, предприятия РКК и т. д. У них нет тематической фильтрации, специализированной маршрутизации, да и вообще архитектура таких решений не рассчитана на то, что данные будут изменяться в зависимости от масштаба карты или фильтров пользователя.

Существуют и более специализированные решения, например, ведомственные карты, интерактивные проекты, связанные с Роскосмосом, или корпоративные интранет-системы, используемые на предприятиях оборонно-промышленного комплекса. Но у них другая проблема: они закрытые, сделаны «для себя» и не предполагают масштабируемости, открытого доступа или использования в образовательных и просветительских целях.

Кроме того, при изучении современных тенденций современные тенденции было выявлено, что многие из них переходят от логики «всё на клиенте» к логике, где сервер берёт на себя основные вычисления — например, кластеризацию и маршрутизацию. Всё чаще применяются гибридные решения, где используются реляционные базы для хранения данных и in-memory хранилища (например, Redis) — для быстрого геопоиска и кеширования. Также всё больше ценится адаптивность: система должна уметь подстраиваться под пользователя, учитывать его интересы, масштаб карты и т. д.

С учётом всего выше сказанного я пришёл к выводу, что разработка собственной системы оправдана. Мне нужно было решение, которое:

- умеет отображать специализированные объекты;
- может масштабироваться и быть интерактивным;
- позволяет строить маршруты по собственным критериям;
- даёт пользователю гибкий инструмент для изучения инфраструктуры ракетно-космической отрасли.

1.5 Применяемые технологии

В процессе проектирования подсистемы и анализа требований к будущей системе был определён набор технологий, на которых строится архитектура программного обеспечения. Выбор каждой технологии основывался на её функциональных возможностях, совместимости с поставленными задачами, производительности, а также широком применении в практике разработки современных веб-сервисов.

1. **JavaScript + TypeScript**

JavaScript является одним из распространённых языков программирования в сфере веб-разработки. Он выполняется прямо в браузере пользователя и используется для создания интерактивных элементов на сайте: от кнопок и выпадающих меню до сложных динамических интерфейсов. Именно благодаря JavaScript современные сайты перестали быть статичными — появилась

возможность мгновенно обновлять контент, взаимодействовать с сервером без перезагрузки страницы, работать с картами, анимациями и пользовательскими данными в реальном времени. В рамках разрабатываемой системы он отвечает за поведение всех элементов на клиентской стороне, таких как карта, интерфейс фильтрации, всплывающие окна и маршруты.

Typescript — это надстройка над JavaScript, которая добавляет в язык статическую типизацию. По сути, это тот же JavaScript, только с дополнительными возможностями контроля кода ещё до запуска. Typescript помогает избежать типичных ошибок, которые сложно отловить в процессе выполнения, так как проверка происходит уже на этапе компиляции. Например, если функция ожидает число, а передаётся строка — компилятор предупредит об этом ещё до запуска приложения. Это особенно удобно в крупных проектах, где важно поддерживать структуру, читаемость и стабильность кода. Однако может возникать проблема с внутренним порогом входа, так как обладает абсолютна своим синтаксисом. Из-за этого неопытный разработчики могут сделать только хуже, из-за того, что будут игнорировать типизацию внутри своего приложения, что после может усложнить поиск и отладку различных проблем. Также он отлично работает с современными редакторами, позволяя получать подсказки, автозаполнение и навигацию по проекту.

Выбор именно этого стека (JavaScript + TypeScript) обусловлен не только его популярностью, но и тем, что он максимально подходит под архитектуру современных веб-приложений. Он позволяет быстро разрабатывать интерфейс, удобно работать с API, обрабатывать пользовательские действия и динамически изменять содержимое страницы. Кроме того, большинство библиотек и фреймворков (например, React, который используется в этом проекте) ориентированы именно на работу с TypeScript.

Такая связка используется практически во всех современных frontend-проектах: от сайтов с погодой до административных панелей и высоконагруженных интерфейсов. В индустрии она применяется такими компаниями, как Google, Microsoft, GitHub и многими другими.

2. React.js

React — это библиотека для разработки пользовательских интерфейсов, разработанная компанией Meta (ранее Facebook). Её основная задача — упростить создание динамических и интерактивных веб-приложений. React помогает разработчикам работать с визуальной частью сайта, разбивая интерфейс на независимые компоненты, которые можно многократно переиспользовать. Это особенно удобно в тех случаях, когда страница постоянно обновляется в зависимости от действий пользователя: например, при фильтрации данных, переключении маршрутов, открытии карточек объектов и т.п.

В основе React лежит концепция **виртуального DOM** — это внутренняя копия дерева элементов интерфейса, с которой библиотека сравнивает текущие изменения перед тем, как отрисовать их в браузере. Благодаря этому React минимизирует количество обращений к реальному DOM, что делает интерфейс отзывчивым и быстрым даже при большом количестве элементов. Это особенно важно в проекте, где на карте могут одновременно отображаться сотни объектов, при этом пользователь взаимодействует с ними в реальном времени.

Ещё одно ключевое преимущество React — это **компонентный подход**. Интерфейс разбивается на небольшие независимые блоки — компоненты: например, кнопка, карточка объекта, модальное окно или панель фильтрации. Каждый компонент может содержать свою собственную логику, стили и внутреннее состояние. Такой подход упрощает разработку, поддержку и масштабирование системы, особенно когда проект становится большим и в нём участвуют несколько разработчиков.

React отлично работает в связке с TypeScript, что позволяет добавить строгую типизацию к компонентам и обеспечить более надёжный код. Также библиотека легко интегрируется с внешними API, что важно в рамках данной

системы, где интерфейс активно взаимодействует с backend-сервисом, получает геоданные, маршруты и фильтрованные результаты.

В промышленной разработке React применяется повсеместно: от интернет-магазинов и новостных платформ до административных панелей, навигационных систем и даже интерфейсов для внутренних корпоративных приложений. Его используют такие компании, как Meta, Netflix, Airbnb, Dropbox, Яндекс и многие другие.

В контексте данной системы React позволяет реализовать удобный, быстрый и гибкий пользовательский интерфейс, который корректно реагирует на действия пользователя, отображает большое количество данных и не требует полной перезагрузки страницы при каждом взаимодействии. Это делает библиотеку отличным выбором для построения фронтенда современной ГИС-платформы.

3. Bun

Bun — это современная JavaScript/TypeScript-среда выполнения, объединяющая в себе функции, которые ранее приходилось реализовывать разными отдельными инструментами. В отличие от привычной связки Node.js + npm + ts-node + Webpack, Bun представляет собой «всё в одном» решение: он одновременно выполняет роль рантайма, менеджера пакетов, сборщика и компилятора TypeScript.

Одной из ключевых причин выбора Bun для реализации серверной части стало его высокое быстродействие. Bun написан на языке Zig и показывает значительно более высокую производительность по сравнению с Node.js при старте, компиляции и выполнении кода. Это особенно актуально в системах с большим количеством мелких запросов и частой компиляцией TypeScript в JavaScript, что как раз характерно для backend-for-frontend решений.

Ещё одним важным преимуществом Bun является его **встроенная поддержка TypeScript «из коробки»** без необходимости настройки дополнительных транспайлеров вроде tsc или babel. Это упрощает структуру проекта и ускоряет процесс разработки и сборки. Также Bun позволяет быстро

запускать тесты, выполнять скрипты и управлять зависимостями через собственный менеджер пакетов bun install, который работает значительно быстрее стандартного npm или yarn.

В рамках данного проекта серверная часть отвечает за:

– обработку REST-запросов от клиентского интерфейса

– обращение к Redis и PostgreSQL;

– выполнение кластеризации объектов;

– кеширование маршрутов и выборку по координатам

– маршрутизацию данных и фильтрацию по пользовательским критериям.

Использование Bun позволяет реализовать всю эту логику с минимальной задержкой, снизить время старта сервера и упростить конфигурацию среды. Это особенно полезно на этапе разработки и локального тестирования, где важна быстрая итерация.

Несмотря на то, что технология относительно новая, она активно развивается и уже применяется в реальных проектах, где важны скорость, лёгкость конфигурации и тесная интеграция с TypeScript. Bun начинает получать широкую популярность в сообществе JavaScript-разработчиков как перспективная альтернатива Node.js в новых проектах.

В контексте данной системы Bun оказался подходящим решением, так как объединил в себе производительность, простоту и гибкость, необходимые для реализации backend-сервиса с высокой частотой запросов и взаимодействием с геоданными.

4. Redis

Redis — это высокопроизводительное хранилище данных, работающее в оперативной памяти. В отличие от классических реляционных баз данных (таких как PostgreSQL), Redis предназначен не для хранения больших массивов структурированных данных, а для быстрого доступа к часто используемой информации. Благодаря тому, что все данные находятся в оперативной памяти, Redis обеспечивает практически мгновенное чтение и запись — задержки

измеряются в миллисекундах, что особенно критично в системах, где важна скорость реакции.

Одна из главных причин использовать Redis в проекте — это его поддержка **географических команд**, таких как GEOADD, GEORADIUS и GEOSEARCH. Эти команды позволяют сохранять координаты объектов и выполнять быстрый геопоиск в заданной области карты. Это особенно важно в задачах, где пользователь постоянно перемещает карту, а система должна мгновенно реагировать и отображать только актуальные данные из текущей видимой области. Благодаря Redis не приходится каждый раз обращаться к основной базе данных, что позволяет значительно снизить нагрузку на сервер и повысить производительность.

Помимо геопоиска, Redis также используется как **кеширующее хранилище**. В системе, где есть часто повторяющиеся запросы (например, построение маршрутов между одинаковыми точками), Redis может сохранять уже готовые результаты и быстро возвращать их при повторном запросе. Это экономит ресурсы и уменьшает время отклика интерфейса. Redis поддерживает механизм TTL (времени жизни ключей), поэтому такие кеши автоматически очищаются спустя заданный промежуток времени, не засоряя память.

Redis хорошо интегрируется с Node.js, что делает его удобным выбором для современных веб-приложений. Он работает с различными структурами данных: списками, хешами, множествами, и, как уже было сказано, с геоданными — что идеально вписывается в задачи данного проекта.

Технология активно используется в индустрии: её применяют GitHub, Twitter, StackOverflow, Twitch, Яндекс, Booking и многие другие крупные компании, которым важно быстро обрабатывать данные в реальном времени. Redis широко применяется в системах, связанных с геолокацией, уведомлениями, подсчётом статистики, а также в качестве промежуточного слоя между клиентом и основной базой.

В контексте данной системы Redis играет роль «ускорителя» — он обеспечивает быструю выдачу данных по геоположению, снижает нагрузку на

PostgreSQL и делает взаимодействие с картой плавным и отзывчивым. Это особенно актуально, когда речь идёт о масштабируемости и большом числе пользователей, одновременно работающих с системой.

5. PostgreSQL + PostGIS (реляционная БД + георасширение)

Почему выбрано: PostgreSQL — надёжная и мощная реляционная база данных с открытым исходным кодом. А расширение PostGIS позволяет выполнять пространственные запросы прямо внутри базы.

Преимущества:

- Строгая типизация и стабильность;
- Продвинутая поддержка геоданных (точки, полигоны, линии, индексы);
- Возможность фильтрации и агрегации данных по координатам;
- Открытое, хорошо документированное решение.

Почему подходит для проекта: Все объекты отрасли (институты, площадки, предприятия) имеют координаты и дополнительную информацию. Система должна уметь искать, фильтровать и группировать объекты по типу, по зоне карты, по признакам — PostgreSQL с PostGIS делает это удобно и быстро.

Кто использует: NASA, OpenStreetMap, QGIS, Министерства транспорта, городские навигационные платформы. PostGIS — один из самых популярных выборов для построения ГИС.

6. Yandex Maps API

Почему выбрано: Yandex Maps API предоставляет доступ к картам, маршрутам и географическим сервисам. Он хорошо подходит для веб-приложений, использующих карты, и особенно удобен для российских геоданных.

Преимущества:

- Детализированные карты и маршруты по РФ;
- Гибкое API для добавления объектов, создания маршрутов и работы с кластерами;

- Поддержка адаптивного отображения и кастомизации карты.

Почему подходит для проекта: Система работает с объектами по территории России, и важно, чтобы картографическая часть была точной и актуальной. Яндекс.Карты обеспечивают высокую детализацию и простоту интеграции в веб-приложение.

Кто использует: Множество российских сервисов, включая транспортные порталы, городские справочники, навигаторы и коммерческие карты.

1.6 Перечень задач, подлежащих решению в процессе разработки

В рамках проектирования и создания информационной системы, предназначенной для отображения и маршрутизации по объектам ракетно-космической отрасли, была сформулирована совокупность задач, которые необходимо решить в процессе разработки. Эти задачи охватывают все ключевые этапы — от анализа предметной области до отладки программных компонентов и подготовки документации.

Задачи, подлежащие решению:

1. **Исследование предметной области**
   - Анализ структуры и особенностей объектов ракетно-космической отрасли;
   - Выявление ключевых типов объектов и их характеристик;
   - Определение логики построения маршрутов между объектами.
2. **Анализ аналогов и существующих решений**
   - Изучение готовых ГИС- и навигационных систем;
   - Оценка применимости существующих решений под специфические требования отрасли;
   - Выявление ограничений аналогов и обоснование необходимости собственной разработки.

3. **Проектирование архитектуры программного обеспечения**
    - Определение общей архитектуры системы: клиент-серверная модель с использованием BFF-подхода;
    - Выделение основных компонентов (frontend, backend, хранилища, API);
    - Разработка логики взаимодействия между частями системы.
4. **Структурирование и подготовка данных**
    - Сбор, очистка и унификация информации об объектах;
    - Классификация по типам, преобразование координат и описание атрибутов;
    - Приведение данных к формату GeoJSON и настройка автоматической конвертации.
5. **Макетирование компонентов пользовательского интерфейса**
    - Проработка структуры карты, модальных окон, фильтров и навигационных элементов;
    - Создание прототипов экранов и логики взаимодействия с геоданными.
6. **Разработка и реализация backend-сервиса (BFF)**
    - Реализация логики работы с Redis и Postgres;
    - Обработка запросов на отображение, фильтрацию и маршрутизацию;
    - Внедрение алгоритма кластеризации (DBSCAN) и маршрутизации на основе Yandex API.
7. **Разработка клиентской части веб-приложения**
    - Отображение карты с геометками;
    - Взаимодействие с BFF-сервисом для подгрузки данных и фильтрации;
    - Построение пользовательских маршрутов и визуализация информации об объектах.
8. **Интеграция компонентов и внутренняя отладка**

- Настройка API-взаимодействия между frontend и backend;
- Проверка корректности передачи данных, отображения и навигации;
- Обработка пограничных сценариев и исключений.

9. **Проведение тестирования и экспериментальной оценки**
   - Измерение производительности при разном количестве объектов;
   - Проверка корректности маршрутов и кластеров;
   - Оценка времени отклика, стабильности системы и обработки пользовательских сценариев.
10. **Оформление технической и пользовательской документации**
    - Описание архитектуры, компонентов, структуры данных и логики работы системы;
    - Подготовка руководства пользователя и инструкций по развёртыванию;
    - Финальное оформление технического задания.