МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки / специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка API сервера для государственных организаций»

Обучающийся С.А. Павлюк _____
(Инициалы Фамилия) (личная подпись)

Руководитель к.ф.-м.н. О.В. Лелонд
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант (ы) к.п.н., доцент, Т.С. Якушева
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2023

Аннотация

Тема бакалаврской работы: «Разработка API сервера для государственных организаций».

Бакалаврская работа посвящена разработке API сервера для государственных организаций.

В ходе выполнения исследований по бакалаврской работе был произведен анализ существующих решений проектирования API, анализ требований, проектирование структуры приложения и реализация приложения.

Во введении прописывается актуальность темы, написаны цель и задачи.

В первом разделе рассматривается предметная область исследования, основные особенности корпоративной разработки, и анализ существующих подходов к проектированию.

Во втором разделе описываются используемые методы и технологии для получения решения, проектирование архитектуры системы и структуры приложения.

Третий раздел содержит реализацию приложения, тестирование и развертывание приложения, и результаты разработки.

В заключении представлены результаты выполнения выпускной квалификационной работы.

Бакалаврской работа состоит из введения, трёх разделов, заключения и списка использованной литературы.

Бакалаврская работа состоит из 43 страниц, 12 рисунков, 1 таблицы, 25 источников и 1 листинга.

Abstract

The title of the bachelor’s thesis is “Development of an API server for government organizations”.

The research is devoted to developing an API server for government organizations.

When doing a research, the analysis of existing API design solutions, requirements analysis, application structure design and application implementation was carried out.

The introduction reveals the relevance of the research and gives a brief description of the work done.

The first section examines the subject area of research, the main features of corporate development, and the analysis of existing approaches to design.

The second section describes the methods and technologies used to obtain a solution, designing the architecture of the system and the structure of the application.

The third section contains the application implementation, application testing and deployment, and development results.

In conclusion, the conclusions of the entire work are drawn.

The bachelor’s thesis consists of an introduction, three sections, a conclusion and list of used literature.

The volume of the bachelor's thesis is 43 pages, it also contains 12 figures, 1 table, a list of 25 references and 1 listing.

Содержание

Введение

Информационные технологии представляют собой постоянно развивающуюся область, играющую значительную роль в модернизации различных отраслей, в том числе государственных организаций [10]. В последние годы государственные организации используют возможности технологий для

улучшения своих услуг, повышения их доступности и эффективности. В этом контексте разработка API-сервера для государственной организации весьма актуальна и служит важным шагом на пути к модернизации.

Работа в соответствующей области обеспечит необходимые технические и управленческие навыки для успешной разработки сервера API для государственной организации.

Предметом исследования является разработка API-сервера, который является важнейшим компонентом современных программных комплексов.

Объектом исследования является государственная организация, которой требуется этот сервер API для оптимизации своих операций, улучшения своих услуг и улучшения связи с другими заинтересованными сторонами.

Целью данной бакалаврской работы является проектирование и разработка сервера API, который облегчит обмен информацией между правительственной организацией и другими заинтересованными сторонами.

Основными задачами этой работы являются:

- анализ требований организации;
- проектирование подходящей архитектуры;
- разработка конечных точек API;
- тестирование и развертывание сервера.

1. Анализ существующих решений проектирования API

1.1 Предметная область

В нынешнюю цифровую эпоху правительственные организации все больше полагаются на технологии для предоставления эффективных услуг своим гражданам. Одним из важных аспектов этого является разработка интерфейсов прикладного программирования (API) для облегчения обмена данными между различными системами. В этом проекте мы предлагаем разработку API-сервера для государственной организации, чтобы упростить ее работу и повысить удовлетворенность граждан.

Основная цель работы – создать сервер API, который может использоваться различными отделами организации для доступа к данным и услугам. Сервер облегчит обмен информацией между различными отделами и поможет им более эффективно работать вместе. Благодаря стандартизированному интерфейсу доступ к данным и их совместное использование осуществляются беспрепятственно и безопасно [15].

Разработка сервера API потребует тщательного планирования и выполнения, чтобы убедиться, что он соответствует требованиям организации. Первым шагом будет определение данных и сервисов, которыми необходимо обмениваться между различными отделами. Это потребует работы с заинтересованными сторонами из разных отделов, чтобы понять их требования и определить ключевые элементы данных, которыми необходимо поделиться.

После определения элементов данных следующим шагом будет проектирование архитектуры сервера API. Это потребует выбора соответствующего стека технологий и разработки общей структуры сервера API. Архитектура должна быть масштабируемой, безопасной и простой в использовании для разработчиков во всей организации.

Следующим шагом будет разработка конечных точек API [2], [24]. Это будут конкретные URL-адреса, которые разработчики могут использовать для доступа к данным и службам с сервера API. Конечные точки должны быть хорошо задокументированы и соответствовать установленным стандартам, чтобы обеспечить простоту их использования и понимания.

В дополнение к разработке сервера API нам также потребуется разработать набор средств разработки программного обеспечения (SDK). Эти SDK будут использоваться разработчиками в организации для взаимодействия с сервером API. SDK должны быть просты в использовании и хорошо документированы, чтобы обеспечить их широкое применение в организации.

На протяжении всего процесса разработки нам нужно будет убедиться, что сервер API безопасен и соответствует всем нормативным требованиям. Это потребует реализации соответствующих мер безопасности, таких как шифрование и аутентификация. Нам также необходимо убедиться, что сервер API соответствует всем правилам конфиденциальности данных и что все конфиденциальные данные защищены.

После того, как сервер API будет разработан, нам нужно будет тщательно его протестировать, чтобы убедиться, что он соответствует требованиям организации. Это будет включать в себя комбинацию модульного тестирования, интеграционного тестирования и пользовательского приемочного тестирования. Нам также необходимо убедиться, что сервер API хорошо задокументирован и что разработчики в организации знакомы с тем, как его использовать.

Разработка сервера API для государственной организации является важным проектом, который может помочь оптимизировать операции и повысить удовлетворенность граждан. Проект будет включать тщательное планирование и выполнение, чтобы гарантировать, что сервер API соответствует требованиям организации и соответствует всем применимым правилам. Разрабатывая стандартизированный интерфейс для доступа к

данным и услугам, мы можем облегчить сотрудничество между различными отделами и повысить общую эффективность.

1.2 Математическое описание

Очереди могут ограничиваться по длине (по числу входящих заявок), заявки могут выполняться как в порядке поступления, так и в любом другом, заранее заданным пользователем. Для случая с MS SQL Server следует использовать одноканальную СМО с ограниченной очередью. Такая система имеет один обслуживающий канал [4]. На вход, потоком, поступают заявки с интенсивностью λ. Если обслуживающий канал занят в тот момент, когда поступает заявка, она встает в очередь и ожидает начала обслуживания. Число мест заранее ограничено и известно. Если обслуживающий канал занят и в очереди нет свободных мест, то входящая заявка уходит из системы не обслуженной. Время обслуживания заявки – это случайная заявка, которая подчиняется экспоненциальному закону распределения с параметром μ. Среднее время обслуживания одной заявки вычисляется по формуле [13], [25]:

$$t = \frac{1}{\mu} \quad (1)$$

где t – среднее время обслуживания одной заявки, μ – интенсивность обслуживания.

Существует несколько возможных состояний системы:

- S_0 канал свободен;
- S_1 канал занят, очереди нет;
- S_{1+1} канал занят, в очереди одна заявка;
- S_{1+2} канал занят, в очереди две заявки;
- S_{1+m} канал занят, в очереди м заявок.

Выделяют следующие показатели эффективности работы СМО:

- вероятность того, что обслуживающий канал свободен:

$$p_0 = \frac{1-p}{1-p^{m+2}} \quad (2)$$

где p – заявка, m – очередь заявок;

– вероятность отказа $p_{\text{отк}}$ (вероятность того, что заявка покинет СМО не обслуженной) $p_{\text{отк}} = p_n$;

– вероятность того, что канал занят, в очереди k заявок:

$$p_{1+k} = p^{1+k} p_0 \quad (3)$$

где k – количество заявок в очереди;

– вероятность отказа (в очереди нет свободных мест):

$$p_{\text{отк}} = p^{1+m} p_0; \quad (4)$$

– относительная пропускная способность Q (отношение среднего числа обслуживаемых в единицу времени заявок к среднему числу поступивших за это время заявок):

$$Q = 1 - p_{\text{отк}}; \quad (5)$$

– абсолютная пропускная способность A (среднее число заявок, которое СМО может обслужить в единицу времени):

$$A = \lambda Q; \quad (6)$$

– среднее число заявок в очереди:

$$L_{\text{оч}} = p^2 \frac{1 - p^m(m+1-mp)}{(1-p^{m+2})(1-p)}; \quad (7)$$

– среднее время нахождения заявки в очереди:

$$T_{\text{оч}} = \frac{1}{\lambda} L_{\text{оч}}; \quad (8)$$

– среднее число занятых обслуживанием каналов:

$$L_{\text{обсл}} = 1 - p_0; \quad (9)$$

– среднее число заявок в системе:

$$L_{\text{сис}} = L_{\text{оч}} + L_{\text{обсл}}; \quad (10)$$

– среднее время нахождения заявки в системе:

$$T_{\text{сис}} = \frac{1}{\lambda} L_{\text{сис}}; \quad (11)$$

1.3 Основные особенности корпоративной разработки

Некоторые из основных особенностей корпоративной разработки для разработки сервера API для государственной организации могут включать [1], [18]:

– сбор требований: понимание конкретных требований и потребностей правительственной организации и ее пользователей;

– выбор технологии: выбор соответствующего стека технологий для разработки сервера API с учетом существующей IT-инфраструктуры организации и будущих требований к масштабируемости;

– принципы проектирования API: разработка четкого и лаконичного дизайна API, соответствующего отраслевым стандартам и передовым методам обеспечения безопасности, масштабируемости и простоты использования;

– прототипирование: разработка функционального прототипа API-сервера для тестирования и уточнения дизайна и функциональности;

– тестирование и обеспечение качества. Проведение обширных испытаний и обеспечение качества для обеспечения того, чтобы сервер API функционировал должным образом и отвечал всем требованиям;

– документация: разработка исчерпывающей документации для сервера API, включая руководства пользователя, документацию для разработчиков и технические спецификации;

– развертывание и обслуживание: развертывание сервера API в рабочей среде и обеспечение текущего обслуживания и поддержки для обеспечения оптимальной производительности и функциональности;

– сотрудничество и общение. Тесное сотрудничество с заинтересованными сторонами государственной организации, IT-отделом и конечными пользователями, чтобы убедиться, что сервер API соответствует их потребностям и ожиданиям.

Для наглядности сделаем схему организации (рисунок 1).

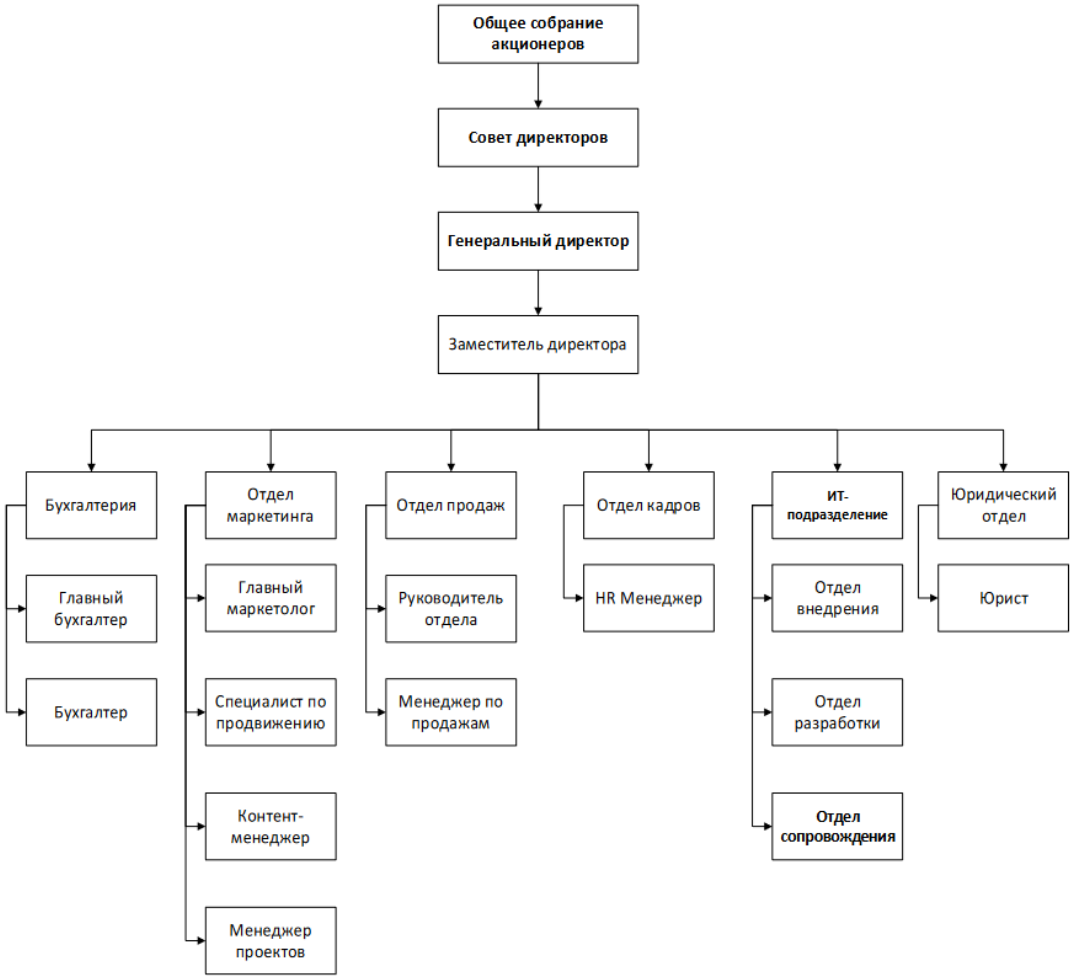

Рисунок 1 – Схема организации

1.4 Анализ существующих подходов к проектированию API

Дизайн API является важным аспектом разработки программного обеспечения, особенно для организаций, которые полагаются на Web-сервисы для предоставления своих продуктов и услуг [3]. В контексте разработки сервера API для государственной организации важно учитывать существующие подходы к проектированию API, чтобы убедиться, что полученный API соответствует потребностям организации и придерживается установленных передовых практик.

Существует несколько существующих подходов к проектированию API, в том числе проектирование на основе моделей, проектирование на основе REST и проектирование на основе гипермедиа. Дизайн на основе модели включает в себя разработку API на основе предварительно определенной модели данных, которая может помочь обеспечить согласованность и ясность дизайна API. Этот подход особенно полезен для организаций, которые имеют хорошо зарекомендовавшие себя структуры данных и системы [8].

Дизайн RESTful, с другой стороны, является более гибким подходом, который фокусируется на разработке API, который следует принципам передачи репрезентативного состояния (REST). REST – это набор архитектурных принципов, которыми руководствуются при проектировании Web-сервисов, уделяя особое внимание простоте, масштабируемости и совместимости с существующими Web-стандартами. RESTful API часто легче внедрять и поддерживать, чем другие типы API, что делает их популярным выбором для многих организаций.

Наконец, проектирование на основе гипермедиа – это более новый подход, который фокусируется на разработке API-интерфейсов с самоописанием и предоставлением ссылок на соответствующие ресурсы. Этот подход подчеркивает важность возможности обнаружения и гибкости, позволяя клиентам более интуитивно и эффективно перемещаться по ресурсам API [9].

При разработке сервера API для государственной организации важно учитывать конкретные потребности и требования организации. В то время как дизайн RESTful является популярным выбором для многих организаций, дизайн на основе моделей может быть более подходящим для организаций с хорошо зарекомендовавшими себя моделями данных и системами. С другой стороны, дизайн на основе гипермедиа может быть хорошим выбором для организаций, которые отдают приоритет доступности и гибкости своих API.

Выбор подхода к разработке API будет зависеть от конкретных потребностей и требований организации [14], [20]. Дизайн на основе моделей,

дизайн RESTful и дизайн на основе гипермедиа – все это допустимые подходы к проектированию API, и лучший подход будет зависеть от потребностей и целей организации. В конечном счете, успех сервера API будет зависеть от тщательного планирования и исполнения, а также постоянного обслуживания и поддержки.

Произведен сравнительный анализ существующих подходов к проектированию API (таблица 1).

Таблица 1 – Анализ существующих подходов к проектированию API

Подход	Описание
GraphQL	Язык запросов для API, который представляет собой более эффективную, мощную и гибкую альтернативу REST. Позволяет клиентам указывать нужные им данные и получать только эти данные, уменьшая сетевой трафик и повышая производительность.
ОТДЫХ	Стиль архитектуры для создания Web-служб, который использует команды HTTP (GET, POST, PUT, DELETE) и URI ресурсов для выполнения операций. Более легкий и гибкий, чем SOAP, но ему может не хватать стандартизации и согласованности.
МЫЛО	Протокол обмена структурированной информацией при реализации Web-сервисов. Использует XML для кодирования сообщений и HTTP для транспорта. Может быть сложным и тяжеловесным.
gRPC	Высокопроизводительная универсальная платформа RPC с открытым исходным кодом, которая использует протокольные буферы для сериализации сообщений и поддерживает несколько языков и платформ. Подходит для создания микросервисов и распределенных систем.

Продолжение таблицы 1

JSON-API	Спецификация для создания API-интерфейсов, которые используют JSON для обмена данными и предоставляют стандартизированные соглашения для отношений ресурсов, разбиения на страницы, фильтрации и обработки ошибок. Помогает обеспечить согласованность и уменьшить двусмысленность.

Выводы по разделу 1

Для проектирования API, была описана предметная область, где пошагово была расписана цель работы. Дальше рассмотрели основные особенности корпоративной разработки для разработки сервера API, где в конце была представлена схема организации.

В конце первого раздела был описан выбор подхода к разработке API и был произведен анализ существующих подходов к проектированию API.

2. Анализ требований и проектирование структуры приложения

2.1 Предъявляемые методы и технологии для получения решения

Для получения решения по разработке сервера API для государственной организации можно представить несколько методов и технологий. К ним относятся [21], [23]:

– дизайн RESTful API является широко принятым стандартом для разработки API. Он включает разработку API таким образом, чтобы они были масштабируемыми, надежными и простыми в использовании. Дизайн RESTful API фокусируется на использовании HTTP-методов для извлечения и обработки данных. Кроме того, он предоставляет единый интерфейс для доступа к API для разных клиентов;

– node.js – это популярная серверная технология, которую можно использовать для разработки сервера API. Он предоставляет быструю, масштабируемую и эффективную платформу для создания серверных приложений. Он также имеет большое сообщество разработчиков и богатую экосистему модулей и пакетов;

– mongoDB – это база данных NoSQL, которую можно использовать для хранения данных для сервера API. Это документно-ориентированная база данных, обеспечивающая масштабируемость, гибкость и производительность. MongoDB также поддерживает распределенные хранилища данных, что делает его идеальным выбором для государственных организаций, которым необходимо хранить большие объемы данных;

– swagger – это платформа с открытым исходным кодом, которую можно использовать для документирования API. Он предоставляет удобный интерфейс для разработчиков, чтобы понять, как использовать API. Swagger также автоматически генерирует клиентский код и серверные заглушки, экономя время и силы при разработке;

– docker – это технология контейнеризации, которую можно использовать для развертывания сервера API. Он предоставляет легкое, портативное и масштабируемое решение для развертывания приложений. Docker также обеспечивает согласованную работу сервера API в различных средах, что упрощает управление и обслуживание.

Эти методы и технологии при совместном использовании могут обеспечить эффективное решение для разработки сервера API для государственной организации [17]. Они обеспечивают масштабируемость, гибкость и производительность, а также простоту использования и обслуживания.

2.2 Используемые методы и технологии для получения решения

Для разработки сервера API для государственной организации могут быть использованы следующие методы и технологии [6], [22]:

– исследование. Проведите тщательное исследование, чтобы понять требования государственной организации и ее заинтересованных сторон. Соберите информацию о существующих системах и процессах и проанализируйте данные, чтобы выявить пробелы, проблемы и возможности;

– дизайн. Разработайте архитектуру, компоненты и рабочие процессы сервера API. Определите API, конечные точки, модели данных и механизмы аутентификации. Используйте такие инструменты, как блок-схемы, диаграммы UML и каркасы, для создания комплексного проекта;

– разработка: разработайте сервер API с использованием языков программирования и сред, таких как Node.js, Python, Flask, Django и Express. Используйте лучшие практики, такие как модульное программирование, контроль версий и проверки кода, чтобы обеспечить надежность, масштабируемость и ремонтопригодность. Используйте такие инструменты, как Postman, Swagger и Charles Proxy, для тестирования и отладки API;

– развертывание. Разверните сервер API на облачной платформе, такой как AWS, Azure или Google Cloud. Используйте инструменты контейнеризации, такие как Docker и Kubernetes, для создания масштабируемой и отказоустойчивой инфраструктуры. Используйте методы DevOps, такие как непрерывная интеграция, доставка и мониторинг, чтобы обеспечить плавное развертывание и эксплуатацию;

– безопасность. Внедрите меры безопасности, такие как шифрование, брандмауэры, обнаружение вторжений и контроль доступа, чтобы защитить сервер API от несанкционированного доступа, утечки данных и кибератак. Используйте рекомендации OWASP и инструменты тестирования безопасности, такие как ZAP и Burp Suite, для выявления и снижения рисков безопасности;

– документация. Создайте исчерпывающую документацию для сервера API, включая справочник по API, руководства пользователя, руководства для разработчиков и руководства по устранению неполадок. Используйте такие инструменты, как Swagger UI и ReDoc, для создания интерактивной и настраиваемой документации;

– обслуживание. Обеспечьте текущее обслуживание и поддержку сервера API, включая исправления ошибок, улучшения функций и оптимизацию производительности. Используйте такие инструменты, как журналы, метрики и оповещения, для мониторинга работоспособности и производительности сервера API. Предоставляйте своевременную и оперативную поддержку пользователям, разработчикам и заинтересованным сторонам сервера API.

А также построим логическую модель базы данных (рисунок 2).

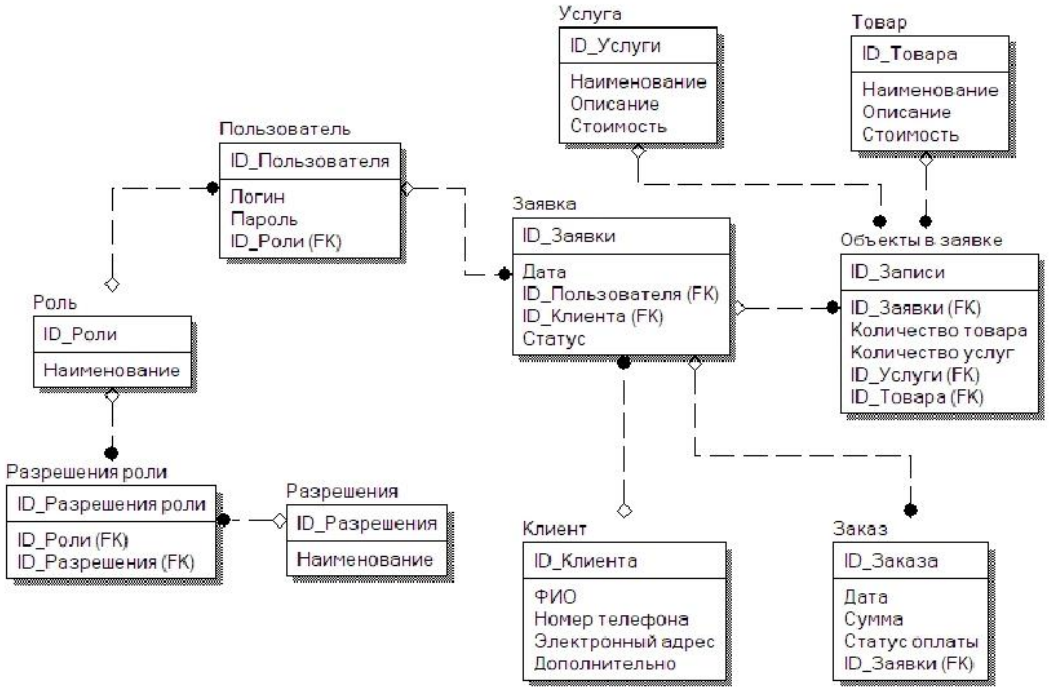

Рисунок 2 – Логическая модель

2.3 Проектирование архитектуры системы

Проектирование архитектуры системы включает следующие шаги:

– определение требований к системе. Анализирование потребностей и требований заказчика, определение задач и функций, которые должна выполнять система, а также конечной аудитории;

– проектирование общей архитектуры. Разработка общей схемы системы, определение ее компонентов и каналов связи между ними, выбор технологий и инструментов;

– проектирование компонентов. Разработка детализированного описания каждого компонента системы и их взаимодействия между собой;

– выбор архитектурных паттернов. Рассмотрение различных архитектурных паттернов, таких как микросервисная архитектура, модульная архитектура, SOA и т.д. и выбор оптимального решения для целей системы;

– выбор базы данных. Выбор системы хранения данных, определение структуры базы данных и ее интерфейса;

– выбор протоколов и алгоритмов. Определение протоколов и алгоритмов, используемых для обмена данными между компонентами системы и для обработки их функций;

– тестирование и оптимизация. Тестирование системы, определение возможных уязвимостей, необходимых исправлений и оптимизации для достижения максимальной производительности.

Ключевую роль в проектировании архитектуры API-сервера для государственной организации играет выбор соответствующей технологической платформы, используемой для разработки и реализации сервера.

В первую очередь необходимо определить функциональные потребности системы, допустимые типы запросов, формат обмена данными, а также требования к безопасности и надежности сервера.

Затем следует выбрать наилучший тип сервера, который может быть монолитичным или модульным (микросервисный), в зависимости от потребностей.

При разработке серверной архитектуры также необходимо обеспечить следующие характеристики: масштабируемость (возможность увеличения размера сервера с увеличением нагрузки), отказоустойчивость, наличие слоя аутентификации и авторизации запросов, применение SSL-шифрования для защиты обмена данными [5].

Важным аспектом является также использование подходов и инструментов, обеспечивающих высокую скорость и эффективность работы сервера, таких как использование кэш-систем, индексации, оптимизации обработки больших объемов данных и т.д [7].

Кроме того, стоит уделить внимание выбору протоколов маршрутизации и общения между клиентом и сервером, таких как REST или GraphQL, а также

реализации контроллеров (обработчиков запросов) и методов API (CRUD, создание, чтение, обновление, удаление данных).

Выбор операционной системы для API-сервера для государственной организации зависит от многих факторов, включая потребности в обработке данных, использование определенных типов хранилищ данных, а также поддержку соответствующих протоколов и инструментов разработки и поддержки сервера.

В общем случае, для API-серверов обычно используются операционные системы Linux или Windows Server [12]. Linux является полностью бесплатной операционной системой с высокой надежностью и безопасностью, а также поддерживает большой спектр языков программирования и инструментов. Windows Server также является популярным выбором для серверных приложений на основе Microsoft.NET и является привычным выбором для пользователей, уже работающих с операционной системой Windows.

Однако, выбор операционной системы также зависит от конкретных требований, таких как поддержка определенных языков программирования или фреймворков, интеграции с существующими системами и инфраструктурой, требований к скорости и производительности, а также возможности масштабирования.

При проектировании архитектуры API-сервера для государственной организации, необходимо учесть множество факторов и принять во внимание конкретные потребности и требования заказчика, чтобы обеспечить максимальную производительность, надежность и безопасность сервера.

При проектировании архитектуры API-сервера для государственной организации, необходимо учесть множество факторов и принять во внимание конкретные потребности и требования заказчика, чтобы обеспечить максимальную производительность, надежность и безопасность сервера (Рисунок 3).

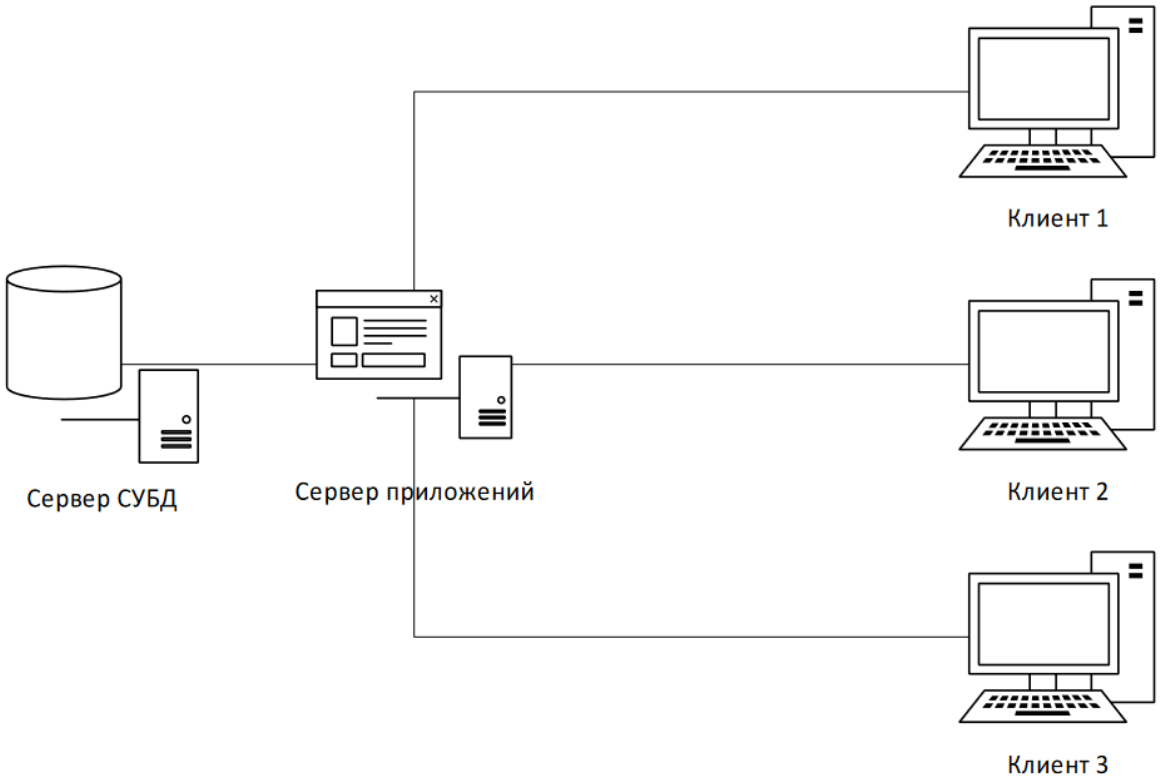

Рисунок 3 – Архитектура системы

2.4 Проектирование структуры приложения

Для проектирования структуры приложения для разработки API сервера для государственной организации следует учитывать особенности безопасности, масштабируемости, производительности и соответствия требованиям законодательства [19]:

– определите требования и цели API, включая типы данных, которые он будет обрабатывать, количество пользователей, которые будут обращаться к нему, а также требования к производительности и безопасности;

– выберите технологический стек (набор технологий), который может поддерживать требования API, такой как определенный язык программирования, фреймворк и базу данных;

– разработайте конечные точки API и форматы данных, рассмотрите возможность использования стандарта, например OpenAPI или Swagger, для документирования API;

– реализуйте механизмы безопасности, такие как контроль доступа, аутентификацию и шифрование, чтобы защитить API от несанкционированного доступа и утечек данных;

– рассмотрите возможность использования балансировщика нагрузки и/или слоя кэширования для улучшения производительности и масштабируемости;

– протестируйте и оптимизируйте API, чтобы обеспечить его соответствие требованиям функциональности и производительности;

– убедитесь в соответствии системы требованиям законодательства (например, GDPR или HIPAA) и примите соответствующие меры для защиты конфиденциальности пользователей.

Конкретика архитектуры приложения для API сервера для государственной организации зависит от многих факторов, таких как размер организации, используемые технологии и конкретные требования к системе. Тем не менее, следующие принципы могут быть использованы при проектировании архитектуры системы [16]:

– разработка микросервисов: поскольку приложение построено на микросервисной архитектуре, его компоненты могут быть разработаны и масштабированы независимо друг от друга. Каждый микросервис может обслуживать только определенную функциональность;

– безопасность данных: данные являются наиболее ценным активом государственных организаций и требуют серьезного уровня безопасности. Поэтому должны быть реализованы механизмы контроля доступа, шифрования и аутентификации;

– обработка ошибок: важно обеспечить обработку ошибок в приложении и предоставить пользователям информацию о проблемах и способы их решения;

– использование популярных технологий: использование популярных технологий, таких как Node.js или Python, может облегчить разработку и обеспечить наличие различных библиотек и плагинов для решения общих проблем;

– мониторинг и журналирование: логирование компонентов приложения может облегчить отладку, а мониторинг уровня обслуживания и производительности позволит определить возможные проблемы и устранить их до того, как они повлияют на работу приложения;

– использование контейнеров: использование контейнерных технологий, таких как Docker, может облегчить управление конфигурацией и развертывание приложений, а также обеспечить удобную масштабируемость;

– непрерывная интеграция и доставка: автоматизация процесса сборки, тестирования и доставки кода позволяет ускорить процесс разработки и снизить количество ошибок, которые могут появиться в коде;

– отказоустойчивость: приложение должно обеспечивать высокую отказоустойчивость, чтобы предотвратить сбои и обеспечить непрерывную работу;

– резервное копирование данных: следует регулярно создавать резервные копии данных, чтобы в случае сбоя можно было быстро восстановить работу приложения;

– соблюдение правил безопасности и законодательства: приложение должно соответствовать принципам защиты данных, а также требованиям законодательства в области информационной безопасности.

Учитывая данные принципы, можно разработать структуру приложения для API сервера государственной организации, которое обеспечит высокий

уровень безопасности, масштабируемость и производительность. Некоторые важные элементы структуры могут включать в себя:

– схема базы данных: определение сущностей и отношений между ними, которые будут использоваться при планировании API и рассмотрении вопросов безопасности и производительности;

– методы аутентификации: выбор самой подходящей методики для обеспечения аутентификации пользователей, включая механизмы контроля доступа и проверки подлинности;

– методы передачи данных: определение форматов и протоколов передачи данных, например, JSON или XML, а также их соответствие внутренним стандартам;

– структура URL: создание простой и интуитивно понятной структуры URL, чтобы пользователи могли легко получить доступ к ресурсам API;

– обработка ошибок: определение методов обработки ошибок, включая возвращение кодов ошибок и сообщений, которые помогут пользователям понять, что произошло;

– мониторинг и журналирование: определение инструментов для мониторинга и журналирования работы сервиса;

– безопасность: реализация мер безопасности, включая авторизацию, аутентификацию, защиту от DDOS атак, а также проверку и обработку входных данных.

Проектирование структуры приложения для разработки API сервера для государственной организации может включать в себя следующие шаги:

– определение требований к API: определение того, что приложение должно делать и какие функции и возможности должны быть доступны через API;

– установка языка программирования: выбор языка программирования, который будет использоваться для написания API сервера.

В общем случае, для API сервера можно использовать любой язык программирования, который поддерживает создание сетевых приложений, например, Python, Java, Node.js, Ruby, PHP и т.д.;

– выбор фреймворков: выбор фреймворков, которые будут использоваться для разработки API сервера. Это может включать в себя фреймворки для web-разработки, например, Flask, Django, Express.js, Ruby on Rails, Symfony и т.д.;

– база данных: определение используемой базы данных, которая будет хранить данные, необходимые для работы приложения. Для государственной организации важно уделять особое внимание безопасности и защите данных;

– методы аутентификации и авторизации: определение методов аутентификации и авторизации, которые будут использоваться для обеспечения безопасности API. В зависимости от поставленных задач, могут использоваться различные методы, такие как OAuth2, JWT и HTTP Basic Authentication;

– API документация: создание документации API, которая содержит информацию о том, как использовать API и какие запросы и ответы ожидать;

– контроль версий: использование системы контроля версий, например, Git, для управления кодом приложения;

– методы передачи данных и форматы: определение форматов, протоколов и методов передачи данных, которые будут использоваться в API. В зависимости от поставленных задач, это может включать в себя форматы JSON, XML, CSV, а также протоколы HTTP, HTTPS, FTP и др.;

– методы обработки ошибок: определение методов обработки ошибок и исключений, которые могут возникнуть в приложении. Важно предусмотреть четкие коды ошибок и сообщения, которые помогут пользователям понимать, что произошло;

– реализация механизмов безопасности: включение мер безопасности, таких как проверка подлинности, шифрование данных и защита от атак;

– мониторинг и журналирование: определение инструментов мониторинга производительности и журналирования работающего приложения, которые помогут выявлять и исправлять проблемы быстро;

– тестирование: определение стратегии тестирования и проведение тестирования API на всех этапах разработки;

– непрерывная интеграция и доставка: автоматизация процесса сборки, тестирования и доставки кода, чтобы обеспечить быструю и надежную доставку приложения в продакшен;

– документация API: создание документации API, которая содержит информацию о том, как использовать API и какие запросы и ответы ожидать, а также примеры кода;

– масштабируемость: создание архитектуры, которая обеспечит возможность масштабирования приложения в будущем.

Для наглядного примера рассмотрим диаграмму вариантов использования и диаграмму компонентов (рисунок 4,5 соответственно).

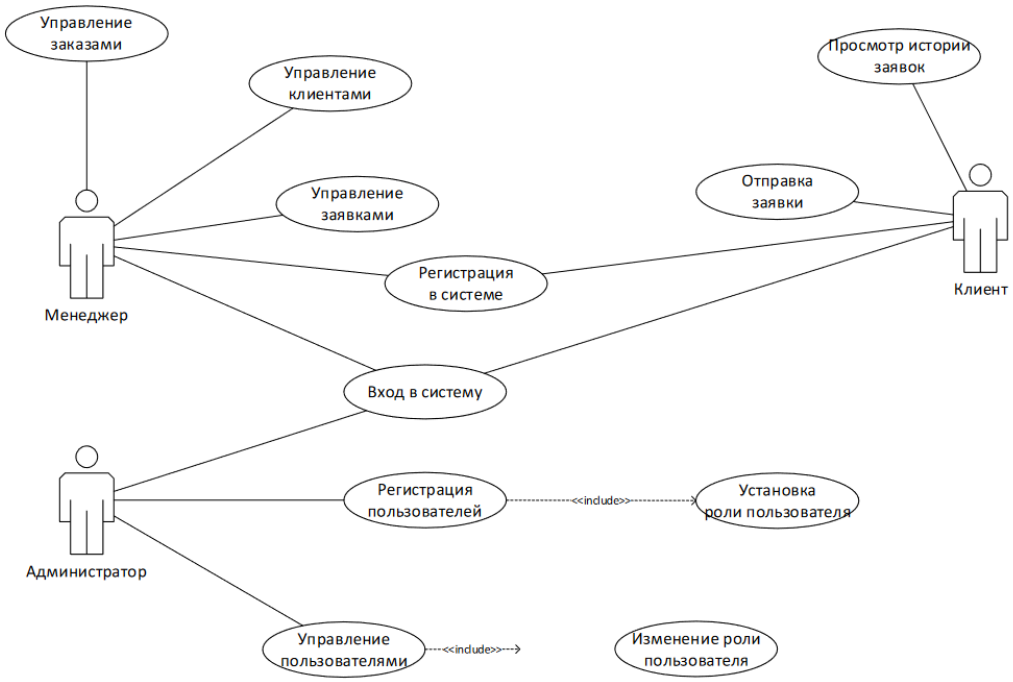

Рисунок 4 – Диаграмма вариантов использования

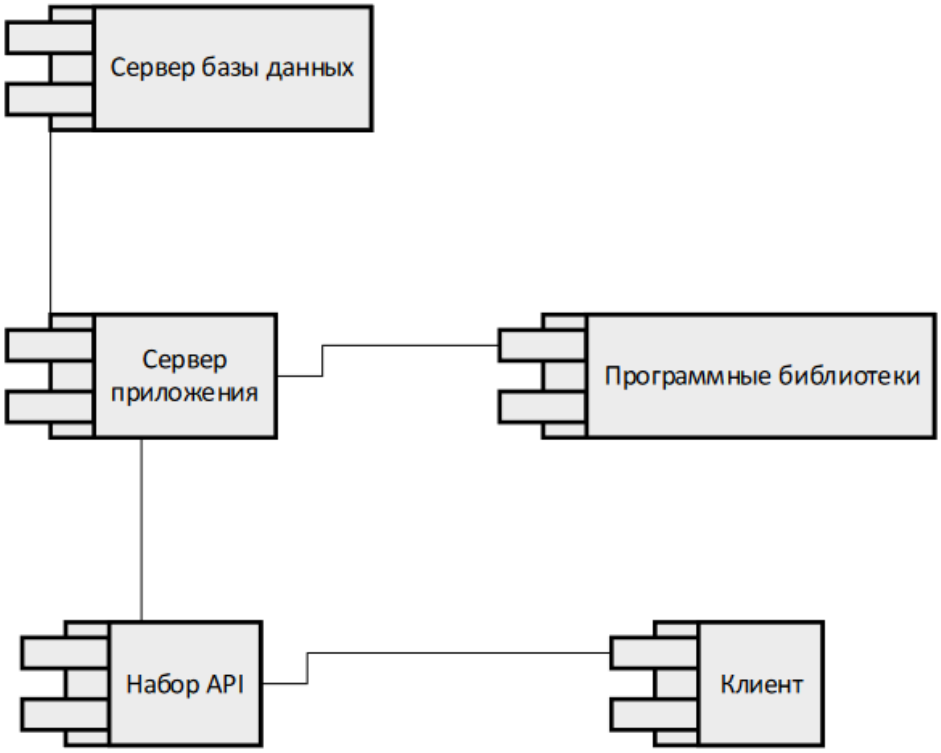

Рисунок 5 – Диаграмма компонентов

Выводы по разделу 2

В этом разделе мы обсудили анализ требований и структуру приложения для нашего решения. Мы предложили различные методы и технологии для получения решения, а также обсудили методы и технологии, которые в конечном итоге были использованы для получения решения.

Кроме того, мы также обсудили дизайн системной архитектуры и то, как структура приложения была разработана для удовлетворения требований. Этот раздел имеет решающее значение, поскольку он закладывает основу для процесса разработки и гарантирует, что окончательное решение соответствует желаемым целям и задачам.

3. Проектная реализация приложения

3.1 Реализация основного приложения

Код основного приложения и вспомогательных служб имеет некоторые сходства, которые были реализованы с помощью системы подмодулей git. Всего есть три общих подмодуля: core, interface и commonApplication. Подмодуль Core управляет моделью и взаимодействием с базой данных, а подмодуль интерфейса предоставляет интерфейсы для работы с этой моделью. Стоит отметить, что взаимодействие классов происходит только на уровне интерфейса, даже для классов модели, что повышает гибкость системы и позволяет заменять определенные классы. Наконец, подмодуль commonApplication содержит общий программный код, такой как промежуточное ПО для обработки запросов (более подробно это будет рассмотрено ниже).

3.1.1 Реализация модели

На этапе проектирования каждому элементу был предоставлен список соответствующих полей и ссылки на дополнительную информацию об этих полях. Однако работа с этими полями в таком формате оказалась неудобной, что потребовало создания интерфейса, извлекающего конкретное поле из его списка на основе его имени или ключа. Кроме того, интерфейс элемента выделяет общие поля, такие как ключ, имя, версия, даты начала и окончания действий, ord, ordinalNumber, ParentKey, IsHasChilds и ChildsCount для иерархических элементов.

После этого был разработан компонент для взаимодействия с базой данных. Этот компонент вызывает процедуры PL/SQL для извлечения данных и возвращает их как средство чтения Oracle [11]. Была добавлена дополнительная абстракция для упрощения процесса доступа к различным

элементам, справочникам, атрибутам и т. д. через компонент взаимодействия с базой данных. Эта абстракция включает в себя маппер, отвечающий за получение данных от считывателя и входных параметров, и возвращает либо список ответов, либо сообщение об ошибке в случае внутренней ошибки во время выполнения процедуры оракула.

Также был создан компонент кэширования данных. Его работа проста: сначала он проверяет, есть ли запрошенные данные уже в Reddis. Если нет, он извлекает данные из компонента базы данных, заполняет кэш Reddis и возвращает из него запрошенные данные.

3.1.2 Схема обработки запросов

После этого разработка компонентов схемы и преобразователя была приоритетной, поскольку они были необходимы для получения данных. Схема состояла из двух типов запросов: mutation и query, которые использовались для получения списка каталогов, из которых можно было генерировать операции создания, удаления, обновления, изменения и получения. Эти операции выполнялись в рамках специализированных распознавателей, основанных на описаниях каталогов. Резолверы использовали ранее существовавшие компоненты, связанные с данными, для получения данных и вызывали конечные точки службы mutation, передавая необходимую информацию для выполнения mutation.

3.1.3 Упаковка данных

Для сортировки элементов используется интерфейс IComparable. Этот интерфейс может преобразовывать поля всех типов (большинство преобразований выполняется автоматически), но для поля ordinalNumber пришлось написать отдельный компаратор из-за его уникального формата. Затем функция сравнения используется для сравнения элементов на основе их

полей. Если сортировка происходит по нескольким полям и есть равенство, устойчивые сортировки вызываются в порядке приоритета, установленного пользователем. Для этого был создан класс ChainComparer. Этот класс принимает IEnumerable IComparer в качестве входных данных и вызывает их в указанном порядке. Кроме того, ChainComparer реализует интерфейс IComparer и может быть установлен для другого объекта.

Листинг 1 – Реализация интерфейса

```
public class ChainComparer<T> : IComparer<T>
   {
      private readonly IEnumerable<IComparer<T>> _comparers;
      public ChainComparer(IEnumerable<IComparer<T>> comparers)
      {
         _comparers = comparers;
      }
      public int Compare(T x, T y)
      {
         foreach (var comparer in _comparers)
         {
            var res = comparer.Compare(x, y);
            if (res != 0)
               return res;
         }
         return 0;
      }
   }
```

Фильтрация включает в себя создание и компиляцию функции из элемента, который возвращает логическое значение. Булева логическая полнота, включающая не, и, и или, в этом случае может быть легко реализована. В зависимости от типа значения существуют различные условия,

такие как «равно», «содержит», «меньше», «больше» (лексикографически) и «не равно» для строковых типов. Кроме того, флаг ignoreCase определяет, следует ли учитывать регистр букв в строке и условии при фильтрации. Для других типов условия равные, меньшие, большие и неравные реализуются на основе их преобразования в интерфейс IComparable и разбора результата вызова метода Compare. И фильтрацию, и сортировку можно выполнять для вложенных полей связанных элементов, при этом процедуры фильтрации или сортировки практически идентичны. Разбиение по страницам может выполняться независимо от структуры данных с помощью операций LINQ skip and take. Версионные элементы также имеют входной аргумент ActualDate для фильтрации версий элементов, действительных на указанную дату. Каждый запрос может отображать дополнительную информацию об элементах, например, их общее количество без фильтрации или количество элементов с фильтрацией (если применяется разбиение по страницам). Для иерархических каталогов было добавлено поле AddParentElements для включения родительских элементов в ответ. Эти параметры делают приложение гибким и предоставляют широкий спектр функций управления данными.

Пример запроса к описанию справочников представлен на рисунке 6.

Помимо CRUD, который является интерфейсом для данных, могут потребоваться определенные вычисления на стороне сервера. Эти вычисления специфичны для поля и подразделяются на отдельные запросы или мутации. Например, каждый орган внутренних дел имеет свой идентификатор, который определяется уездом, в котором он находится. Хотя таких частных запросов или мутаций существует множество, они связаны с предметной областью государственной автоматизированной системы правовой статистики и не имеют. не будут обсуждаться в этой статье, так как они не относятся непосредственно к API для данных с динамически изменяющейся структурой. Как полезный помощник, я надеюсь, что это объяснение понятно.

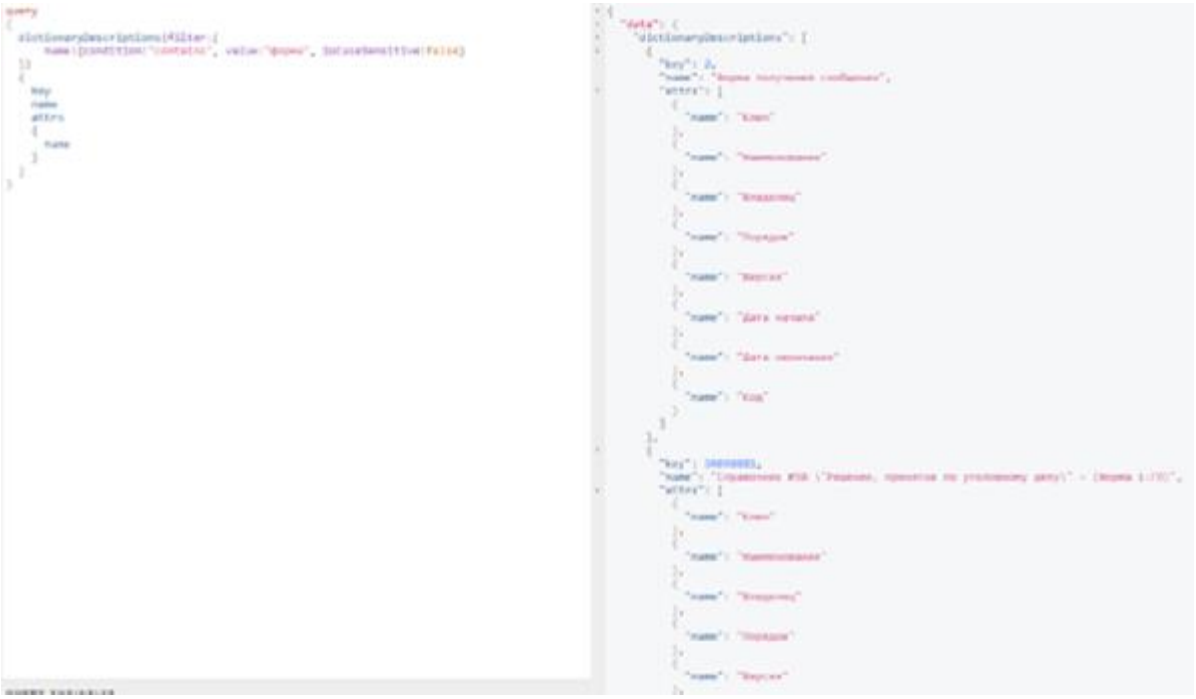

Рисунок 6 – Запрос описания справочников

Кроме того, было обнаружено, что используется значительное количество словарей, что потребовало создания дополнительного объекта для их организации на стороне пользователя. Таким образом, был создан объект папки, который ведет себя аналогично папке файловой системы, позволяя содержать вложенные папки и словари. Чтобы облегчить их управление, были сформулированы отдельные преобразователи и реализованы те же функции фильтрации, сортировки и пейджинга.

Пример запроса к папкам и содержащихся в ним описаниям словарей изображен на рисунке 7.

Пример запроса к элементам справочника изображен на рисунке 8.

Примеры были сгенерированы с помощью сервиса GraphiQL, приложения с открытым исходным кодом, предназначенного для написания запросов к приложениям GraphQL. Кроме того, это приложение использовалось для тестирования новой функциональности перед созданием пост-запроса в Postman.

Рисунок 7 – Запрос папок справочников

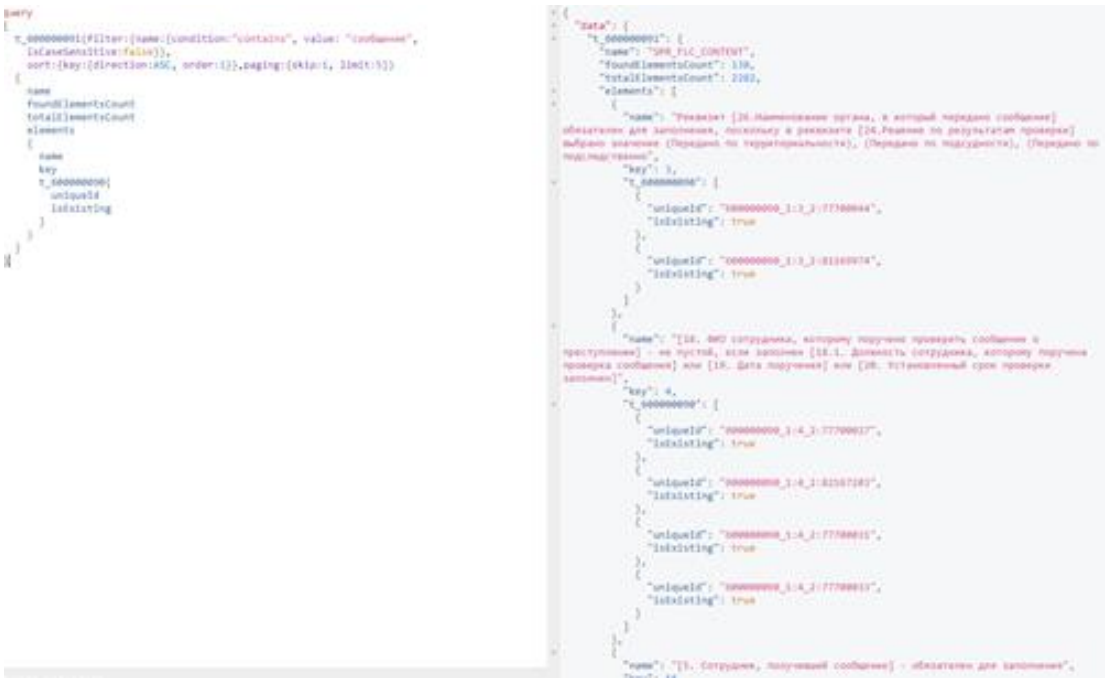

Рисунок 8 – Запрос элементов справочника

Диаграмма кода основного приложения изображена на рисунке 9. По ней видно, что количество связей в приложении достаточно мало. Это достигается за счет того, что все классы взаимодействуют исключительно через интерфейсы.

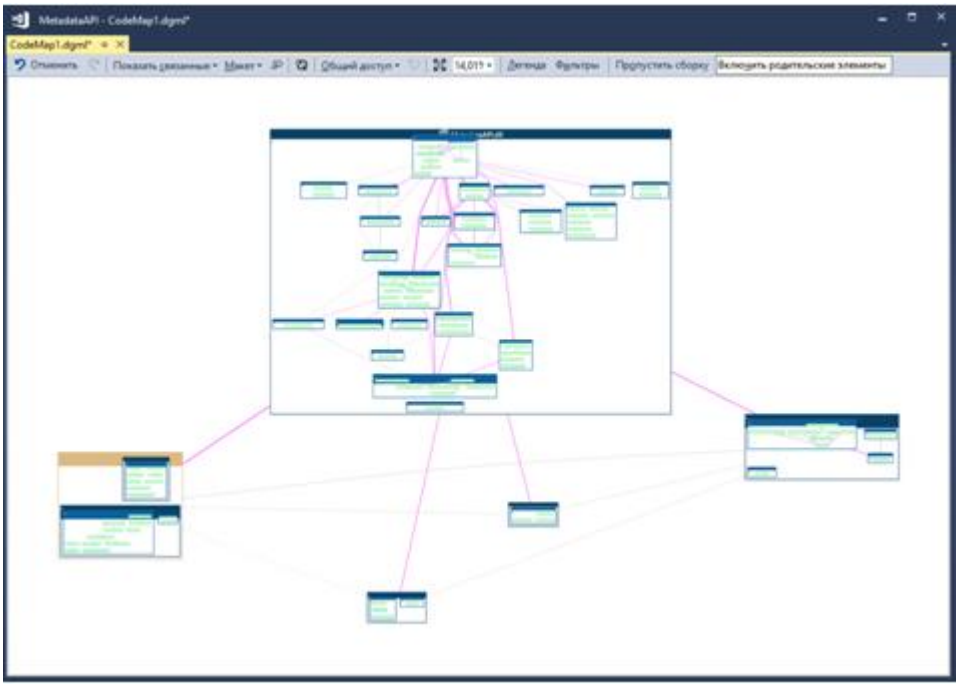

Рисунок 9 – Диаграмма кода основного приложения

3.2 Тестирование и развертывание

Тестирование системы можно поделить на четыре части:

1. Модульное тестирование.
2. Endpoint тестирование.
3. Интеграционное тестирование.
4. Ручное тестирование.

Фреймворк XUnit и принцип инверсии зависимостей используются для модульного тестирования, что позволяет тестировать отдельные части кода отдельно с использованием фиктивных зависимостей. Тестирование конечной точки также использует XUnit, но тестирует большую часть кода, за

исключением вызовов внешних служб и баз данных, достигаемых за счет реконфигурации контейнера IoC. Интеграционное тестирование, которое проверяет всю систему в ее среде, было прекращено во время разработки, чтобы уменьшить зависимости и разрешить публикацию отдельных сервисов. Ручное тестирование проводится для того, чтобы убедиться, что новая функциональность соответствует ожиданиям клиентов, а также для учета любых отзывов клиентов посредством формулировки тестовых примеров. Jenkins используется для запуска всех тестов, за исключением ручного тестирования и интеграционного тестирования.

В ВКР использовалась методология git-flow, при этом ветка dev выступала в качестве рабочей ветки, а ветка master представлялась заказчику. Обновления из ветки dev добавляются в ветку master после того, как команда подтвердит их стабильность и корректную работу. Тесты проводятся, чтобы гарантировать отсутствие регрессии даже перед публикацией в ветке dev. В результате системные службы, включая базу данных и клиент, существуют в двух версиях: dev и master. Кроме того, были включены два варианта без аутентификации в целях тестирования и отладки. В этих неаутентифицированных вариантах отсутствует служба аутентификации, но есть служба GraphiQL. Этот сервис позволяет тестировать и отправлять запросы на языке GraphQL к основному сервису и предоставляет примеры запросов к основному сервису GraphQL.

Архитектура системы без аутентификации изображена на рисунке 10, а архитектура системы с аутентификацией изображена на рисунке 11.

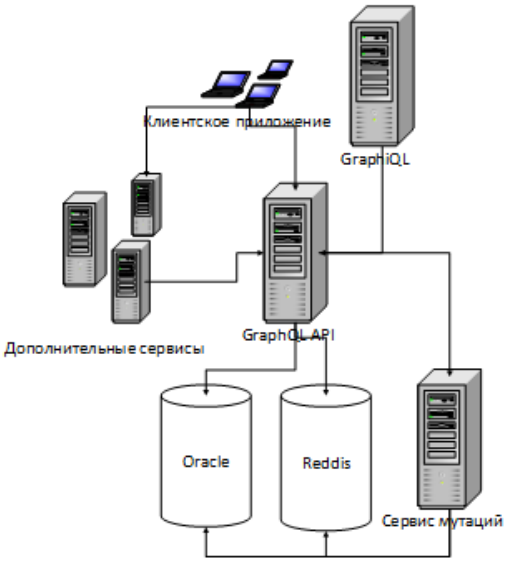

Рисунок 10 – Архитектура системы без аутентификации

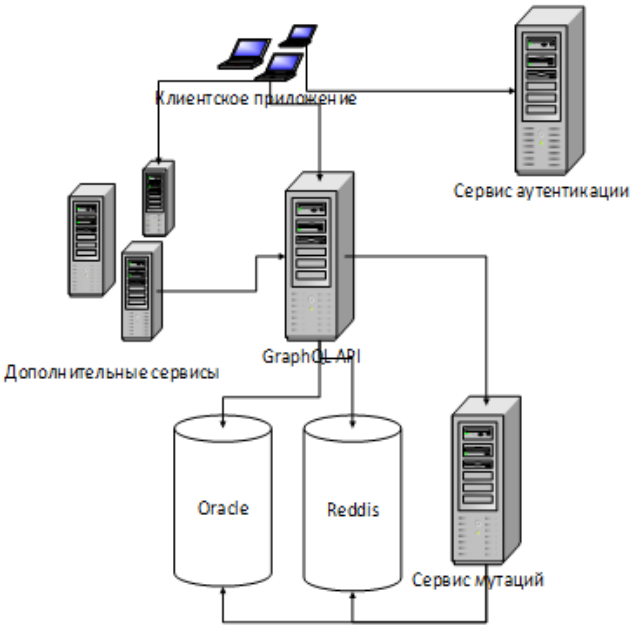

Рисунок 11 – Архитектура системы с аутентификацией

Для обеспечения гибкости развертывания реализован настраиваемый конвейер .NET Core. На этот конвейер можно повлиять во время запуска приложения на основе файла конфигурации, что устраняет необходимость в перестроении приложения.

Конвейер обработки запросов изображен на рисунке 12.

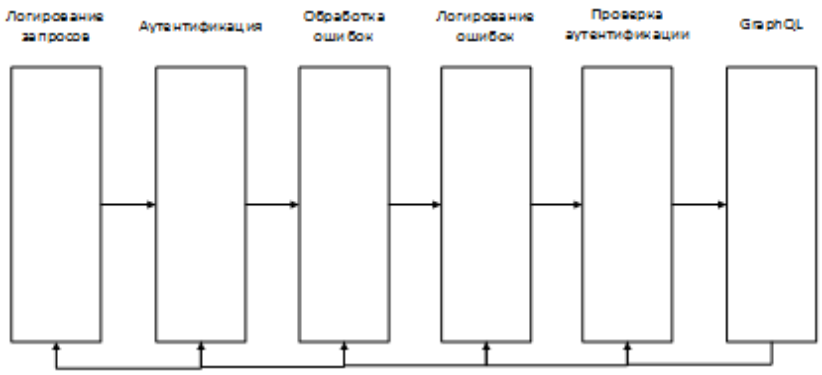

Рисунок 12 – Конвейер обработки запроса

Следует отметить, что разные слои в сервисе можно отключить или заменить другими. Например, когда служба находится на стадии разработки, обработка ошибок вернет всю информацию об ошибке, например трассировку стека. Однако, когда он находится на стадии мастера, предоставляется только информация о пользователе. Если конфигурация не требует аутентификации, уровень проверки аутентификации можно удалить. Для достижения этой замены используется комбинация функций IoC, таких как контейнер .net core и конвейер.

Более того, с помощью конвейера разработан единый протокол взаимодействия между сервисами, построенный поверх HTTP. Этот протокол позволяет восстанавливать состояния ошибок в службе, где они изначально возникли, позволяя ошибкам подниматься вверх по протоколу вплоть до вызова первой операции на стороне клиента.

3.3 Результаты разработки

В результате этой работы было разработано динамическое приложение GraphQL API, предоставляющее доступ к данным с метаданными на основе словарей, содержащих различные поля. Метаданные можно преобразовать,

изменив структуру словаря, а для словарей и их элементов реализован контроль версий. Основная цель приложения – помощь органам внутренних дел Российской Федерации в сборе статистических данных о преступлениях и правовой системе.

Кроме того, сгенерированный GraphQL API используется различными сервисами. Первая служба – это служба экспорта, которая позволяет экспортировать динамические данные в форматы xml/xlsx/rar на основе структуры словаря. Второй сервис компилирует файлы Javascript для экранов автоматов, используемых для сбора или обработки данных из этой системы. Компиляция файлов сценариев Javascript следует правилам для различных полей и экранов путем сбора различных правил из базы данных в функционирующий код Javascript.

Крайне важно установить четкий и единый протокол связи между различными службами. Таким образом, система использует протокол на основе запросов POST, который предоставляет подробную информацию об ошибках, источниках и дополнительную информацию.

Данная система сервисов предоставляет API для интерфейса государственной автоматической системы правовой статистики. Кроме того, хорошо организованная кодовая база и система автоматического обновления облегчают обслуживание. Система также включает модульные и интеграционные тесты, которые выявляют деградацию системы и повышают уверенность в ее надежности. [14].

Выводы по разделу 3

В результате этой работы было разработано динамическое приложение GraphQL API, предоставляющее доступ к данным с метаданными на основе словарей, содержащих различные поля. Кроме того, сгенерированный GraphQL API используется различными сервисами.

Заключение

Бакалаврская работа посвящена разработке API сервера для государственных организаций.

Для проектирования API была описана предметная область, где пошагово расписывалась цель работы. Дальше рассмотрели основные особенности корпоративной разработки для сервера API, с последующим представлением схемы организации.

В конце первого раздела описан выбор подхода к разработке API и произведен анализ существующих подходов к проектированию API.

Для разработки сервера API для государственной организации была создана логическая модель, описана и реализована архитектура системы. Спроектирована структура приложения, где для наглядного примера рассмотрены диаграмма вариантов использования и диаграмма компонентов.

Описана и разработана автоматизированная система управления событиями безопасности на языке программирования Python и фреймворке Django.

В результате этой работы разработано динамическое приложение GraphQL API, предоставляющее доступ к данным с метаданными на основе словарей, содержащих различные поля. Кроме того, сгенерированный GraphQL API используется различными сервисами.

Задачи, определённые для достижения цели работы, выполнены в полном объёме, а именно:

- проанализированы требования организации;
- спроектирована подходящая архитектуры;
- разработаны конечные точки API;
- протестирован и развернут сервер.

Цель работы была достигнута, был разработан API сервер для государственных организаций.

Список используемой литературы

1. Антонов К. О модели взаимодействия клиент-сервер простыми словами. Архитектура «клиент-сервер» с примерами // ZametkiNaPolyah.ru. 2016. URL: https://zametkinapolyah.ru/servera-i-protokoly/o-modeli-vzaimodejstviya-klient-server-prostymi-slovami-arxitektura-klient-server-s-primerami.html (дата обращения: 01.02.2023).
2. Дизайн GraphQL-схем // Github.com. 2019. URL:https://github.com/nodkz/conf-talks/tree/master/articles/graphql/schema-design (дата обращения: 01.02.2023).
3. Мартишин, С.А. Проектирование и реализация баз данных в СУБД MySQL с использованием MySQL Workbench: Методы и средства проектирования информационных систем и техноло / С.А. Мартишин, В.Л. Симонов, М.В. Храпченко. - М.: Форум, 2018. - 61 с.
4. Поллис, Г. Разработка программных проектов на основе Rational Unified Process(RUP) / Г. Поллис, Л. Огастин, К. Лоу. – М.: Бином-Пресс, 2009. – 346 с.
5. Построение Enterprise-приложения // Java-course.ru. 2019. URL: http://java-course.ru/student/book2/scheme/ (дата обращения: 01.02.2023).
6. Репин В. В. Бизнес-процессы. Моделирование, внедрение, управление / В. В. Репин. – М.: Манн, Иванов и Фербер, 2013. 512 с.
7. Степанов Д.Ю. Проблемы внедрения корпоративных информационных систем: уровень приложений // Менеджмент сегодня. 2018. URL: http://stepanovd.com/science/30-article-2015-1-erpappl (дата обращения: 01.02.2023).
8. Шёнталер, Ф. Бизнес-процессы. Языки моделирования, методы, инструменты / Ф. Шёнталер. - М.: Альпина Паблишер, 2019. – 264 с.
9. Яргер, Р.Дж. MySQL и mSQL: Базы данных для небольших предприятий и Интернета / Р.Дж. Яргер, Дж. Риз, Т. Кинг. - М.: СПб: СимволПлюс, 2015. - 560 с.

10. AGILE – гибкая система управления проектами // 4brain.ru. 2017. URL: https://4brain.ru/blog/agile/ (date of accessed: 01.02.2023).

11. Driving commerce to the Web – Corporate Intranets and the Internet: Bitbucket. Gitflow Workflow // Atlassian.com. 2019. URL: https://ru.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow (date of accessed 01.02.2023).

12. Driving commerce to the Web – Corporate Intranets and the Internet: Microsoft. System Namespace // Microsoft.com. 2019. URL: https://docs.microsoft.com/ru-ru/dotnet/api/system?view=netframework-4.7.2. (date of accessed 01.02.2023).

13. Driving commerce to the Web – Corporate Intranets and the Internet: Oracle. Procedures and Packages // Oraclle.com. 2019. URL: https://docs.oracle.com/cd/A97630_01/appdev.920/a96590/adg10pck.htm (date of accessed 01.02.2023).

14. Eltaeib, T., Venna, T.-V.-S.-N., Madasu, V. SOLID Principles in Software Architecture and Introduction to RESM Concept in OOP. Journal of Engineering Science and Technology, 2015. p. 18-25.

15. Fateev D.S., Klochkov K.S., Saburova V.V. Class Design Principles. Russian Federation: Young Scientist, 2016. p. 175-177.

16. Fielding R. Architectural Styles and the Design of Network-based Software Architectures//Ics.uci.edu. 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (date of accessed 01.02.2023).

17. Gutsch, J., Driving commerce to the Web – Corporate Intranets and the Internet: DZone. GraphiQL for ASP.NET Core // Dzone.com. 2017. URL: https://dzone.com/articles/graphiql-for-aspnet-core (date of accessed 01.02.2023).

18. Hartig, O., Pérez, J. An Initial Analysis of Facebook's GraphQL Language, 2017. p. 20-27.

19. Kalinichenko, G.A., Skorokhod, S.V. Comparison of GraphQL and REST API technologies in the development of modern costumer-server applications, Russian Federation: Internauka, 2017. p. 47-52.

20. Martin, R. The Clean Coder: A Code of Conduct for Professional Programmers. Russian Federation: Peter, 2017. p. 153-187.

21. Rest-api [Электронный ресурс] / Режим доступа: https://mcs.mail.ru/blog/vvedenie-v-rest-api (дата обращения: 19.05.2023).

22. Rest-api [Электронный ресурс] / Режим доступа: https://www.ibm.com/ru-ru/cloud/learn/rest-apis (дата обращения: 19.05.2023).

23. Symfony [Электронный ресурс] / Режим доступа: https://symfony.com (дата обращения: 18.05.2023).

24. The technological benefits of Symfony in 6 easy lessons [Электронный ресурс]/symfony.com. – Режим доступа: http://symfony.com/sixgood-technical-reasons (дата обращения: 09.05.2023).

25. Zurmo [Электронный ресурс] / Режим доступа: https://hellip.com/ru/product/zurmo.html (дата обращения: 18.05.2023).