|  |  |  |  |
| --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  «Национальный исследовательский ядерный университет «МИФИ»  Озерский технологический институт -  филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский ядерный университет «МИФИ»  (ОТИ НИЯУ МИФИ) | | | |
| Кафедра прикладной математики | | | |
| Допускается к защите  И. о. зав. кафедрой  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Ф. Зубаиров  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г. | | | |
| Курсовая работа | | | |
| по дисциплине: | Базы данных | | |
| Тема работы: | Автоматизированная система аптечного учета | | |
| Пояснительная записка  МИФИ.092131.102ПЗ | | | |
|  | |  |  |
| Руководитель | |  | Е.Н. Сёмин |
|  | |  |  |
| Н. контролер | |  | И.А. Тарасова |
|  | |  |  |
| Выполнил | |  |  |
| студент гр. 1ИВТ-31Д | |  | К.А. Шабалин |
|  | |  |  |
|  | |  |  |

2024

|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  «Национальный исследовательский ядерный университет «МИФИ»  Озерский технологический институт -  филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский ядерный университет «МИФИ»  (ОТИ НИЯУ МИФИ) |

ЛИСТ ОЦЕНКИ КУРСОВОЙ РАБОТЫ

|  |  |
| --- | --- |
| Студент | Шабалин Кирилл Андреевич |
| Группа | 1ИВТ-31Д |
| Дисциплина: | Базы данных |
| Тема работы: | Преимущества использования GraphQL в разработке веб-приложений на примере интернет-магазина |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Компонент оценки | | | | | Максимальное  значение (баллы) | | Оценка  (баллы) |
| Оценка руководителя за работу в семестре | | | | | 40 | |  |
| Содержание пояснительной записки | | | | | 20 | |  |
| Презентация работы и защита | | | | | 20 | |  |
| Ответы на дополнительные вопросы | | | | | 20 | |  |
| Итого баллов: | | | | | 100 | |  |
| Итого оценка: | A (отл)  90-100 | B (хор)  85-89 | C (хор)  75-84 | D (хор)  70-74 | D (уд)  65-69 | E (уд)  60-64 | F (неуд)  ниже 60 |

Члены комиссии:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Фамилия И. О. |  | Подпись |
|  |  |  |
| Фамилия И. О. |  | Подпись |
|  |  |  |
| Фамилия И. О. |  | Подпись |
|  |  |  |
| Фамилия И. О. |  | Подпись |

|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  «Национальный исследовательский ядерный университет «МИФИ»  Озерский технологический институт -  филиал федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский ядерный университет «МИФИ»  (ОТИ НИЯУ МИФИ) |

ЗАДАНИЕ НА КУРСОВОЕ ПРОЕКТИРОВАНИЕ

|  |  |  |
| --- | --- | --- |
| Дисциплина: | Базы данных | |
| Тема работы: | Автоматизированная система аптечного учета | |
| Дата выдачи задания: | | 25.02.2024 |
| Дата защиты работы | | 06.06.2024 |

КРАТКОЕ ОПИСАНИЕ ЗАДАНИЯ

Разработать автоматизированную систему управления аптечным учётом, включающую базу данных на PostgreSQL и клиент-серверное веб-приложение. Система должна обеспечивать контроль товарных запасов, управление персоналом, учёт клиентов, поддерживать программы лояльности и акции, а также автоматизировать процессы закупки и поставки медикаментов. Проект должен включать детальное проектирование и реализацию базы данных на PostgreSQL, серверной части на Java Spring и клиентской части с использованием JavaScript. Необходимо также разработать механизм защиты данных от несанкционированного доступа и провести тестирование созданной системы для обеспечения её надёжности и производительности.

Аннотация

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | К.А. Шабалин. Автоматизированная система аптечного учета: Курсовая работа. ОТИ НИЯУ МИФИ, 2024, 60 стр., 36 ил.  Библиография - 7 наименований.  Вложения - CD-диск с демоверсией. | | | | | |
| В курсовой работе разработана автоматизированная система аптечного учета, направленная на повышение эффективности и точности работы аптек и аптечных сетей. Проект охватывает все этапы от анализа требований и проектирования базы данных до разработки клиент-серверного приложения и его тестирования.  Предлагается использовать:  а) СУБД PostgreSQL для надежного и масштабируемого управления данными;  б) Java Spring для создания безопасного и высокопроизводительного клиент-серверного приложения;  в) среду разработки IntelliJ IDEA для удобной и эффективной разработки, отладки и тестирования кода;  г) инструменты PgAdmin или DBeaver для дополнительно удобства и возможностей при работе с базой данных, включая выполнение запросов и анализ данных.  Эти предложения позволят разработать автоматизированную систему аптечного учета, основываясь на современных принципах проектирования и безопасности. | | | | | | | | | | | |
|  |  |  |  |  | МИФИ.092131.102ПЗ | | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № докум. | Подпись | Дата |
| Разраб. | | Шабалин |  |  | *Автоматизированная система аптечного учета* | |  | | | Лист | Листов |
| Пров. | | Сёмин |  |  | К |  |  | 4 | 60 |
|  | |  |  |  | ОТИ 1ИВТ-31Д | | | | |
| Н.контр. | | Тарасова |  |  |
| Утв. | | Зубаиров |  |  |

Лит.

Содержание

Введение 6

1 Предметная область 9

1.1 Общая характеристика 9

1.2 Требования к хранимым данным 13

1.3 Специфика и функции 15

1.4 Ограничение и требования к базе данных 16

1.5 Роли в системе 17

1.6 Выбор технологий для серверной части 18

1.7 Бизнес-ограничения 19

2 Проект логической структуры БД 22

2.1 Определение атрибутов и функциональных зависимостей 22

2.2 Нормализация 23

3 Описание общего алгоритма работы системы 26

4 Практическая часть 28

4.1 Описание веб-приложения 28

4.2 Описание серверной части 32

5 Экспериментальная часть 39

5.1 Тестирование ограничений целостности 39

5.2 Тестирование ограничений бизнес-логики 42

6 Информационная безопасность 46

7 Установка и эксплуатация 49

Заключение 56

Список использованных источников 58

# Введение

Разработчики все чаще обращаются к GraphQL как к альтернативе традиционному REST API, так как он предоставляет более гибкий и оптимизированный подход к запросам данных. В условиях постоянно растущего объема данных и разнообразия требований к функциональности веб-приложений, использование GraphQL позволяет снизить нагрузку на сервер, оптимизировать передачу данных и улучшить взаимодействие клиента с сервером, что особенно актуально для интернет-магазинов, где скорость и точность обработки данных критически важны.

Цель данной курсовой работы — продемонстрировать преимущества использования GraphQL в разработке веб-приложений на примере интернет-магазина, реализованного с использованием React и Spring. Для достижения этой цели необходимо решить следующие задачи:

- Провести теоретический анализ особенностей GraphQL и его отличий от традиционного REST API.

- Разработать прототип интернет-магазина, реализующего взаимодействие клиента с сервером через GraphQL.

- Провести анализ преимуществ использования GraphQL для веб-приложений, таких как снижение объема данных, повышение скорости загрузки и улучшение пользовательского опыта.

Объектом исследования является разработанное веб-приложение для интернет-магазина, в котором используется GraphQL для взаимодействия между фронтендом и бэкендом. Предметом исследования выступает практическое применение GraphQL для создания API, его влияние на производительность и структуру взаимодействия компонентов приложения.

Методы исследования включают теоретический анализ существующих подходов к построению API, разработку прототипа интернет-магазина с использованием GraphQL, а также сравнительный анализ преимуществ и особенностей этого подхода в сравнении с традиционными методами, такими как REST API.

## 1 Теоретические основы и преимущества GraphQL

## 1.1 История создания GraphQL

GraphQL был разработан в Facebook в 2012 году как внутренний инструмент для оптимизации работы с API. В то время компания сталкивалась с растущими проблемами масштабирования и эффективной передачи данных между клиентами (мобильными приложениями и веб-страницами) и серверами. Традиционные REST API, используемые Facebook, не всегда обеспечивали нужную гибкость и скорость, а это было особенно важно для мобильных приложений, где ограниченность трафика и высокая отзывчивость имеют решающее значение.

Основная проблема заключалась в том, что REST API возвращали избыточные данные. Например, для отображения информации на ленте новостей в мобильном приложении зачастую приходилось получать несколько видов данных (пользовательские профили, изображения, комментарии и т.д.), каждый из которых требовал отдельного запроса к REST API. Это вело к многочисленным перегрузкам сети, особенно когда приложение запрашивало данные по медленным или дорогим соединениям, таким как мобильные сети.

В ответ на эти проблемы в 2012 году Facebook создал GraphQL как язык запросов, позволивший клиентам гибко формировать запросы и получать только ту информацию, которая им необходима. GraphQL дает клиенту возможность указать конкретные поля и вложенные объекты, которые нужны в данный момент, а также избежать лишних запросов к серверу. Это стало важным шагом к созданию более оптимизированных и гибких API, где нагрузка на сеть и сервер уменьшается, а клиент получает данные быстрее и удобнее.

В 2015 году Facebook сделал GraphQL проектом с открытым исходным кодом (open-source), что привлекло к нему широкое внимание разработчиков и компаний по всему миру. Этот шаг способствовал популярности GraphQL и его внедрению в разнообразные проекты, от стартапов до крупных корпоративных решений. С тех пор вокруг GraphQL была сформирована целая экосистема библиотек, инструментов и серверных реализаций на различных языках программирования, таких как JavaScript (Apollo), Python (Graphene), Java (включая Spring GraphQL) и других.

## 1.2 Место GraphQL в веб-разработке

GraphQL быстро занял свое место в веб-разработке как мощная альтернатива традиционному REST. В то время как REST API структурированы вокруг различных ресурсов, каждый из которых требует отдельного запроса, GraphQL позволяет запросить разные виды данных одним запросом, что значительно упрощает работу с данными для сложных клиентских приложений. Важно понимать, что GraphQL не замещает REST, но предоставляет альтернативный подход для тех случаев, где REST не так удобен или эффективен.

Наибольшее распространение GraphQL получил в следующих областях:

- Мобильные приложения: при работе с мобильными устройствами важно минимизировать объем данных, чтобы экономить трафик и увеличивать скорость загрузки. С GraphQL мобильные приложения могут запрашивать только необходимые данные и не тратить время на получение ненужной информации.

- Single Page Applications (SPAs): в таких приложениях часто требуется динамически обновлять данные и обращаться к API для отображения пользовательского интерфейса. GraphQL, позволяющий делать один запрос на получение данных из разных источников, идеально подходит для сложных интерфейсов, где необходимо обновлять множество связанных данных.

- Микросервисная архитектура: в микросервисных системах данные распределены между разными сервисами. GraphQL позволяет объединять данные из разных источников и предоставлять их клиенту в одном запросе, что значительно упрощает интеграцию между микросервисами и клиентским приложением.

## 1.3 Принципы работы GraphQL: схема, запросы, мутации, подписки.

GraphQL — это мощный инструмент для работы с данными, который предоставляет клиентам большую гибкость в получении необходимых данных. В отличие от традиционных REST API, где существуют жестко определенные конечные точки, GraphQL позволяет клиентам указывать точный формат и объем данных, которые они хотят получить.

### 1.3.1 Схема (Schema)

Схема GraphQL определяет структуру данных, с которыми API может работать. Это своего рода "контракт" между клиентом и сервером, описывающий все возможные типы данных, запросы, мутации и подписки, доступные в API. Схема строится из следующих элементов:

- Типы объектов (Object Types): например, User, Post, Comment.

- Типы запросов (Query Types): определяют доступные для выполнения операции запросы, например, getUser или getPosts.

- Типы мутаций (Mutation Types): описывают операции изменения данных, такие как createUser, updatePost, deleteComment.

- Типы подписок (Subscription Types): позволяют клиентам получать обновления в реальном времени.

Схема обычно описывается на языке SDL (Schema Definition Language), который является частью спецификации GraphQL. Пример схемы:

type User {

id: ID!

name: String!

email: String!

}

type Query {

user(id: ID!): User

allUsers: [User]

}

type Mutation {

createUser(name: String!, email: String!): User

}

type Subscription {

userCreated: User

}

Рисунок

### 1.3.2. Запросы (Queries)

Запросы — это основной способ получения данных в GraphQL. В отличие от REST, клиент сам выбирает поля, которые он хочет получить. Например, клиент может запросить информацию о пользователе, указав только те поля, которые ему нужны:

query {

user(id: "1") {

name

email

}

}

Рисунок

В ответ сервер вернет JSON с запрошенными данными, без лишней информации:

{

"data": {

"user": {

"name": "John Doe",

"email": "john.doe@example.com"

}

}

}

Рисунок

### 1.3.3. Мутации (Mutations)

Мутации используются для создания, обновления или удаления данных. Как и в случае с запросами, клиент может указывать только нужные поля в ответе. Например, запрос на создание нового пользователя с возвратом его идентификатора и имени:

mutation {

createUser(name: "Jane Doe", email: "jane.doe@example.com") {

id

name

}

}

Рисунок

Ответ сервера может выглядеть так:

{

"data": {

"createUser": {

"id": "2",

"name": "Jane Doe"

}

}

}

Рисунок

### 1.3.4. Подписки (Subscriptions)

Подписки позволяют клиентам получать обновления в реальном времени. Например, если приложение хочет получать уведомления о создании новых пользователей, оно может использовать подписку userCreated. Подписки обычно используются вместе с протоколом WebSocket для поддержки постоянного соединения.

Пример подписки:

subscription {

userCreated {

id

name

}

}

Рисунок

Когда сервер генерирует событие, подписанные клиенты получают данные в реальном времени:

{

"data": {

"userCreated": {

"id": "3",

"name": "Alice"

}

}

}

Рисунок

## 1.4 Проблемы REST API и как их решает GraphQL

REST API, долгое время считавшийся стандартом для построения API, обладает рядом недостатков, которые становятся особенно заметными при разработке современных клиент-серверных приложений. Рассмотрим ключевые проблемы REST API и то, как GraphQL помогает их преодолевать.

### 1.4.1 Избыточность и недостаточность данных

При работе с REST API часто возникает проблема избыточности или недостаточности данных. Это происходит из-за фиксированных конечных точек, которые возвращают жестко определенные наборы данных. Например, если клиенту нужны только имя и электронная почта пользователя, но конечная точка GET /user/{id} возвращает всю информацию о пользователе, клиент получит лишние данные, которые ему не нужны. Эта проблема называется избыточностью данных.

Обратная ситуация, недостаточность данных, возникает, когда клиенту приходится отправлять несколько запросов к разным конечным точкам для получения полной информации. Например, для отображения данных о пользователе и его последних постах клиенту может потребоваться сначала запросить данные пользователя, а затем отправить второй запрос для получения постов.

GraphQL решает эти проблемы благодаря своей гибкости. Клиенты могут запрашивать только те данные, которые им нужны, с точностью до полей. В одном запросе клиент может запросить всю необходимую информацию из разных объектов, избегая избыточности и недостаточности данных.

query {

user(id: "1") {

name

email

posts(limit: 5) {

title

content

}

}

}

Рисунок

В результате клиент получает только те данные, которые ему нужны, минимизируя лишнюю нагрузку и снижая объем передаваемой информации.

### 1.4.2 Производительность и количество запросов

REST API может оказаться неэффективным, когда клиенту нужно выполнить несколько запросов для получения всех необходимых данных. Это увеличивает задержку и снижает производительность приложения, особенно на мобильных устройствах или в условиях ограниченного интернета. Большое количество запросов к серверу также увеличивает нагрузку на сеть и может приводить к перегрузке серверов.

GraphQL позволяет объединять данные из разных источников в одном запросе, что снижает количество запросов к серверу. Клиенту достаточно одного запроса, чтобы получить данные, которые в REST API требовали бы нескольких запросов. Это упрощает обработку данных и делает приложение более отзывчивым.

### 1.4.3 Самодокументируемость и типизация API

REST API требует наличия отдельной документации, и разработчики часто сталкиваются с проблемой ее актуальности. Кроме того, в REST API может отсутствовать строгая типизация данных, что приводит к ошибкам на клиентской стороне.

Схема GraphQL API самодокументируется: она предоставляет полный список доступных типов, запросов и мутаций, а также их возвращаемые типы. Это значительно упрощает работу разработчиков, поскольку они могут увидеть все доступные поля и их типы непосредственно через API, используя, например, инструменты GraphQL Playground или GraphiQL. Строгая типизация также помогает отловить ошибки еще на этапе написания кода, обеспечивая его стабильность и предсказуемость.

### 1.4.4 Упрощение работы с данными на клиентской стороне

В REST API клиент должен самостоятельно управлять состоянием данных, которые он запрашивает и обновляет. Это приводит к увеличению сложности и объема кода на клиентской стороне.

GraphQL предоставляет механизм подписок, который позволяет автоматически получать обновления данных в реальном времени. Это особенно полезно для приложений, требующих динамического обновления интерфейса, таких как социальные сети или чаты. GraphQL позволяет клиенту получать новые данные сразу после изменения на сервере, что значительно упрощает логику на клиентской стороне.

# 2 Разработка интернет-магазина с использованием GraphQL