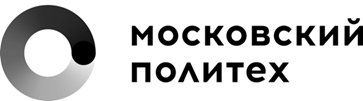
Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(МОСКОВСКИЙ ПОЛИТЕХ)



ЛАБОРАТОРНАЯ РАБОТА 6

**ПРОЕКТИРОВАНИЕ API. ЯЗЫК ЗАПРОСОВ GraphQL.**

По курсу

**Проектирование Веб-сервисов**

ВыполнилКилеев С.И**.**

**студент группы 211-321**

Проверил

Пардаев А.А.

Москва, 2024

**Лабораторная работа 6**

**Цель работы:** Получить представление об основных концепциях GraphQL, таких как схемы, типы, запросы и мутации.

**Задачи:**

Индивидуальная работа

1. Описать в каких случаях применение GraphQL API является целесообразнее.  
2. Разработать схему GraphQL API по выбранной функции из лабораторной работы 4.  
3. Создать таблицы в PostgreSQL или ресурсе Hasura (hasura.io) из лабораторной работы 4.  
4. Составить несколько (не менее трех) запросов и мутаций к полученному GraphQL API.

**Отчёт по выполнению:**

Применение GraphQL API может быть целесообразнее в следующих случаях:

1. **Избыточность данных в REST API**:
   * Если REST API возвращает слишком много данных, которые клиенту не нужны, или приходится делать несколько запросов для получения всех необходимых данных. В GraphQL клиент сам указывает, какие данные ему нужны, что снижает объем передаваемой информации.
2. **Гибкость запросов**:
   * В REST API структура ответа обычно жёстко привязана к конечным точкам, а в GraphQL клиент может выбирать, какие поля ему нужно получить. Это особенно полезно для сложных приложений, где требуется получать разные наборы данных в зависимости от контекста.
3. **Оптимизация запросов**:
   * GraphQL позволяет избежать множества запросов на сервер (Overfetching и Underfetching), так как можно комбинировать данные из разных источников в одном запросе. Это особенно важно в случаях, когда необходимо объединить несколько источников данных в одном запросе.
4. **Удобство для фронтенд-разработчиков**:
   * Фронтенд-разработчики могут самостоятельно запрашивать именно те данные, которые им необходимы, без необходимости изменения бекенда при изменении требований на фронте.
5. **Сложные взаимосвязи между данными**:
   * Если ваши данные имеют сложные взаимосвязи (например, вложенные структуры или отношения), GraphQL упрощает работу с такими данными, позволяя удобно запрашивать и возвращать вложенные структуры.
6. **Реализация подписок (Subscriptions)**:
   * GraphQL поддерживает подписки на события в реальном времени, что полезно в приложениях, требующих мгновенного обновления данных (например, чатах, уведомлениях).
7. **Единая точка доступа к разным источникам данных**:
   * GraphQL может использоваться как единая точка доступа к различным базам данных или микросервисам, объединяя их в один API.

Функция: Оценивать полезность ответов “лайками”

Функциональные требования:

Отобразить оценку ответа: на входе мы получаем ответ на вопрос, а на выходе отображаем количество “лайков” над ответом

Добавить “лайк”: на входе мы получаем нажатую кнопку “лайка” и отрицательную проверку наличие лайка над ответом, а на выходе добавляем единицу от текущего значения “лайков”.

Убрать “лайк”: на входе мы получаем нажатую кнопку “лайка” и положительную проверку наличие лайка над ответом, а на выходе убавляем единицу от текущего значения “лайков”.

Схема:

type Commentary {

commentarty\_id: ID!

user\_id: ID!

topic: String!

content: String!

count\_likes: Int!

theme\_id: ID!

timestamp: String!

message\_type: Boolean!

reply\_message\_id: ID

tag\_id: ID

}

type Query {

commentaries: [Commentary!]!

}

type Mutation {

addLike(commentarty\_id: Int!): Commentary!

removeLike(commentarty\_id: Int!): Commentary!

}

Реализация:

1.

query {

commentaries{

count\_likes

}

}

2.

mutation {

removeLike(commentarty\_id: 1){

topic

count\_likes

}

}

3.

mutation {

addLike(commentarty\_id: 1){

topic

count\_likes

}

}