**8. Поясните понятие GraphQL.**

GraphQL — это язык запросов для API, а также среда выполнения для выполнения этих запросов с использованием существующих данных. Он был разработан Facebook и позволяет клиентам точно указывать, какие данные им нужны, а серверу — только эти данные предоставлять. В отличие от REST, где сервер определяет структуру ответов, в GraphQL клиент сам указывает, какие поля ему нужно вернуть, что делает запросы более гибкими и оптимизированными.

**Основные особенности:**

* **Запросы:** Клиенты могут запрашивать только те данные, которые им нужны, что позволяет уменьшить объем передаваемых данных.
* **Мутации:** GraphQL поддерживает не только чтение данных, но и их изменение (мутации).
* **Подписки:** GraphQL поддерживает обновления в реальном времени через подписки.

**9. Поясните понятие схема GraphQL.**

Схема в GraphQL описывает структуру доступных данных и операций, которые могут быть выполнены на сервере. Она состоит из типов данных, запросов, мутаций и подписок. Все операции, которые могут быть выполнены, а также все типы данных, должны быть заранее определены в схеме.

**Элементы схемы:**

* **Типы (Types):** Описывают структуру данных (например, объект User с полями name, email).
* **Запросы (Queries):** Описывают операции чтения данных.
* **Мутации (Mutations):** Описывают операции изменения данных.
* **Подписки (Subscriptions):** Обеспечивают возможность подписки на обновления данных.

Пример:

type Query {

getUser(id: ID!): User

}

type User {

id: ID

name: String

email: String

}

**10. Расшифруйте аббревиатуру SDL GraphQL.**

**SDL** расшифровывается как **Schema Definition Language**, что означает язык определения схемы. Это текстовый формат для описания схемы GraphQL, который позволяет задавать типы данных, запросы, мутации и подписки. SDL используется для определения структуры данных, с которыми можно взаимодействовать через API GraphQL.

Пример SDL:

type Query {

getUser(id: ID!): User

}

type User {

id: ID

name: String

}

**11. Поясните понятие resolver GraphQL.**

**Resolver** — это функция, которая отвечает за выполнение операций по запросу данных. Когда клиент запрашивает данные через GraphQL, резолверы определяют, как эти данные извлечь и вернуть. Они могут получать данные из различных источников: базы данных, внешних API и т.д.

Каждый поле в запросе GraphQL связано с резолвером, который отвечает за получение или вычисление соответствующих данных. Например, если клиент запрашивает пользователя, резолвер получит пользователя из базы данных и вернёт его.

Пример резолвера:

const resolvers = {

Query: {

getUser: async (parent, args, context) => {

return await User.findById(args.id);

}

}

};

**12. Поясните понятие query GraphQL.**

**Query** в GraphQL — это операция для получения данных с сервера. Клиенты используют запросы (queries) для получения информации в нужном формате и с нужными полями. Запросы описывают, какие данные клиент хочет получить от сервера.

Пример query:

query {

getUser(id: "123") {

name

email

}

}

Этот запрос запросит данные пользователя с ID "123" и вернёт его имя и email.

**13. Поясните понятие mutation GraphQL.**

**Mutation** — это операция, которая позволяет изменять данные на сервере, то есть выполнять CRUD-операции (создание, обновление, удаление). В отличие от запроса (query), который только извлекает данные, мутация изменяет данные.

Пример mutation:

mutation {

createUser(name: "John", email: "john@example.com") {

id

name

email

}

}

Этот запрос создаст нового пользователя и вернёт его ID, имя и email.

**14. Поясните понятие subscription GraphQL.**

**Subscription** — это механизм в GraphQL, который позволяет клиенту подписаться на события на сервере. Когда происходит определённое событие (например, изменение данных), сервер отправляет обновления подписавшимся клиентам. Это позволяет создавать приложения с функциональностью в реальном времени.

Пример subscription:

subscription {

newMessage {

text

sender

}

}

Этот запрос будет слушать новые сообщения и получать их в реальном времени, когда сервер отправит обновление.

**15. Поясните понятие context GraphQL.**

**Context** — это объект, который передаётся ко всем резолверам во время выполнения запроса. Он содержит общую информацию, которая может быть полезна для выполнения запросов, например, информацию о пользователе (аутентификация), настройки сессии или подключения к базе данных.

Пример использования контекста:

const resolvers = {

Query: {

getUser: async (parent, args, context) => {

if (!context.user) throw new Error('Unauthorized');

return await User.findById(args.id);

}

}

};

Здесь контекст может содержать информацию о текущем пользователе, которая будет проверяться перед выполнением запроса.

**16. Поясните схему работы модуля GraphQL.**

Модуль GraphQL работает следующим образом:

1. **Запрос от клиента:** Клиент формирует запрос (query, mutation или subscription) и отправляет его на сервер.
2. **Парсинг запроса:** Сервер принимает запрос, парсит его и проверяет, что запрос соответствует схеме.
3. **Вызов резолверов:** Для каждого поля, указанного в запросе, вызываются соответствующие резолверы.
4. **Получение данных:** Резолверы выполняют необходимые операции (например, получение данных из базы данных).
5. **Формирование ответа:** После получения данных из резолверов, сервер формирует ответ в виде JSON и отправляет его клиенту.

**17. Поясните следующие компоненты схемы GraphQL: subscription, interface, enum, fragment, union.**

* **Subscription:** Это механизм для подписки на события, которые происходят на сервере. Клиент получает обновления в реальном времени. Обычно используется для приложений с функциональностью реального времени, таких как чаты или уведомления.

Пример:

subscription {

newMessage {

text

sender

}

}

* **Interface:** Интерфейсы в GraphQL описывают общие поля, которые должны быть реализованы типами, которые его наследуют. Это позволяет создавать более гибкую иерархию типов.

Пример:

interface Animal {

name: String

}

type Dog implements Animal {

name: String

breed: String

}

* **Enum:** Enum (перечисление) — это тип данных, который ограничивает значения определённым набором. Он используется для предсказуемых данных, например, для полей, которые могут принимать только несколько фиксированных значений.

Пример:

enum Status {

ACTIVE

INACTIVE

PENDING

}

* **Fragment:** Фрагменты — это повторно используемые части запросов, которые могут быть использованы для выборки одинаковых данных в нескольких частях одного запроса.

Пример:

fragment userDetails on User {

name

email

}

query {

getUser(id: "123") {

...userDetails

}

}

* **Union:** Union — это тип, который может быть одним из нескольких типов. Это позволяет возвращать несколько различных типов данных из одного поля, например, когда результат запроса может быть разным типом в зависимости от контекста.

Пример:

union SearchResult = Book | Author

type Query {

search(text: String!): SearchResult

}

Каждый из этих компонентов используется для повышения гибкости и удобства работы с GraphQL.