(Part 3)

API Design

Software System Design

Spring 2024 @ Ali Madooei

GraphQL API Design

GraphQL is a query language for APIs and a runtime for executing those queries.

- **Developed by Facebook:** To address the needs for more efficient and powerful data fetching in modern applications.
- **Declarative Data Fetching:** Clients specify what data they need, avoiding over-fetching or under-fetching of data.
- **Single Endpoint:** All requests are directed to a single endpoint, improving maintainability and organization of API interactions.

Key Concepts of GraphQL

- **Schema:** Defines types, queries, and mutations, acting as a contract between client and server.
- **Resolvers:** Functions that determine how data for a specific field is fetched and returned to the client.
- **Querying:** Allows clients to specify needed data, reducing over or under-fetched data.
- **Mutations:** Operations that create, modify, or delete data, defined specifically in the schema.
- **Subscriptions:** Enable real-time data updates, allowing clients to receive data as it changes.

GraphQL Schema

The schema is the blueprint of your GraphQL API; it defines types, queries, and mutations representing the API's capabilities.

```
type Query {
  post(id: ID!): Post
}

type Mutation {
  createPost(input: PostInput): Post
}

type Post {
  id: ID
  title: String
  body: String
  author: User
}

// Other type definitions...
```

GraphQL Query

In GraphQL, queries are used to read or fetch values.

```
fetch('/graphql', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Accept': 'application/json',
  },
  body: JSON.stringify({
    query: `{
      post(id: "1") {
        title
        author {
          name
        }
        comments {
          body
        }
      }
    }`
  })
}).then(response => response.json());
```

GraphQL Mutation

Mutations in GraphQL are used to create, update, and delete data.

```
// Client Side
fetch('/graphql', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Accept': 'application/json',
  },
  body: JSON.stringify({
    query: `mutation {
      createPost(input: { title: "New Post",
        id
        title
        body
      }
    }`
  })
}).then(response => response.json());
```

GraphQL Subscriptions

Subscriptions in GraphQL enable real-time data updates, allowing clients to receive data as it changes.

GraphQL’s rsubscriptions is enabled through WebSockets or other real-time communication protocols.

```
// Client Side
const ws = new WebSocket('ws://localhost:400
ws.onopen = () => {
  ws.send(JSON.stringify({
    type: 'subscription',
    payload: {
      query: `subscription {
        newPost {
          id
          title
          body
        }
      }`
    }
  }));
};
```

When to Use GraphQL?

- For complex systems with interrelated data, where clients need to fetch nested or related resources in a single request.
- When reducing the number of requests and minimizing over-fetching or under-fetching of data is important.
- For APIs that need real-time data updates (using subscriptions).
- When the API schema is frequently changing or evolving.

GraphQL API: Activity

- Which of the three use cases (University Student Hub App, Boom Virtual Meeting Service, To-Do App) would be best suited for a GraphQL API?

GraphQL API: Activity Solution

- **Student Hub App:** The app would benefit from using a GraphQL API, as it requires complex queries and real-time updates, and needs to fetch nested or related resources in a single request.

- **To-Do App:** Using a GraphQL API is okay, but we do not need complex queries and real-time updates which could be better handled by RESTful APIs. Moreover, caching of GraphQL responses is not as straightforward as RESTful APIs.

- **Boom Virtual Meeting Service:** While a GraphQL API could be used for this service, it may be more efficient to use other API types due to requirements such as low-latency communication and bi-directional streaming capabilities, which can be better accommodated by those API types.

GraphQL API Design: Best Practices

- **Define a Clear and Concise Schema:** Acts as a contract between client and server, representing the API’s capabilities.
- **Optimize for the Client:** Enable clients to request exactly what they need, avoiding over-fetching or under-fetching.
- **Use Descriptive Names:** Choose field and operation names that clearly convey their purpose and functionality.
- **Error Handling:** Provide clear and informative error messages.

GraphQL API: Activity

1. Identify the types and fields that would be included in the GraphQL schema for the Student Hub app.
2. Describe the role of resolvers in the GraphQL API for the Student Hub app.
3. Provide examples of queries and mutations that would be useful for the Student Hub app.
4. Discuss how subscriptions could be utilized in the Student Hub app for real-time updates.

GraphQL API: Activity Solution

1. **GraphQL Schema**:
   - Types: `Event`, `Club`, `User`, etc.
   - Fields for `Event`: `id`, `title`, `date`, `location`, `description`.
   - Fields for `Club`: `id`, `name`, `category`, `description`.

2. **Resolvers**: functions that connect the fields in the GraphQL schema to the data sources.
   - For the Student Hub app, resolvers would handle fetching events, clubs, and user information from the database or external APIs.

GraphQL API: Activity Solution (Contd.)

3. **Queries Example**:
   - `events`: Retrieve a list of events.
   - `club(id: ID!)`: Retrieve details of a specific club.
4. **Mutation Example**:
   - `updateEvent(id: ID!, title: String, date: String, location: String, description: String)`: Update an event's details.

GraphQL API: Activity Solution (Contd.)

4. **Subscriptions**: can be used for real-time updates, such as notifying users when a new event is added or when there are updates to an existing event.
    - For example, a subscription `` `eventAdded` `` could be used to notify users when a new event is created.

GraphQL vs RESTful APIs

- **Data Retrieval:** GraphQL allows clients to request exactly the data they need, while REST retrieves all data associated with the resource that leads to over-fetching or under-fetching.
- **Number of Requests:** GraphQL can retrieve all needed data in a single request, while REST may require multiple requests to different endpoints.
- **Schema and Type System:** GraphQL uses a strong type system and defines the capabilities of the API using a schema, ensuring more reliable interactions.
- **Cacheability:** RESTful APIs are more cacheable due to the use of standard HTTP methods and headers.

GraphQL vs REST APIs (Contd.)

- **Use GraphQL When:**
  - Diverse, nested, and complex data requirements exist.
  - The frontend team needs flexibility in data retrieval.
- **Use REST When:**
  - The API has simple, flat, and static data structures.
  - A standard, cacheable, and stateless API is suitable.

GraphQL Concept Check: Activity

1. **Question 1:** Does GraphQL operate over HTTP?
2. **Question 2:** Is GraphQL considered RESTful?

GraphQL Concept Check: Activity Solution

1. **Answer 1:** Yes, GraphQL does operate over HTTP, typically using a single endpoint to handle all interactions. (With the exception of subscriptions, which use WebSockets or other real-time communication protocols.)

2. **Answer 2:** No, GraphQL is not considered RESTful. It falls short on requirements of uniform interface, statelessness, and cacheability, which are the key principles of RESTful APIs.