Great 🚀 Let's continue. Here's the next one:

---

18. GraphQL with Node.js

GraphQL is a **query language for APIs** and a runtime to execute those queries. Unlike REST (multiple endpoints), GraphQL exposes a **single endpoint** where clients request only the data they need.

---

1. Why GraphQL over REST?

- **REST Issues**: Over-fetching (extra data) & under-fetching (multiple requests).
- **GraphQL Advantage**:
  - Ask for exactly what you need.
  - One endpoint `/graphql`.
  - Strongly typed schema.
  - Better for frontend apps (React, Vue, etc.).

---

2. Core Concepts

1. **Schema** → Defines data types & relationships.

```
type User {
  id: ID!
  name: String!
  email: String!
}
```

2. **Queries** → Read data.

```
query {
  users {
    id
    name
  }
}
```

3. **Mutations** → Modify data.

```
mutation {
  addUser(name: "John", email: "john@test.com") {
    id
```

```
      name
    }
  }
```

4. **Resolvers** → Functions that handle queries/mutations.

```
const resolvers = {
  Query: {
    users: () => db.getUsers()
  },
  Mutation: {
    addUser: (_, { name, email }) => db.addUser(name, email)
  }
};
```

---

3. GraphQL in Node.js with Apollo Server

📌 **Setup Example:**

```
npm install apollo-server graphql
```

```
const { ApolloServer, gql } = require("apollo-server");

// Schema
const typeDefs = gql`
  type User {
    id: ID!
    name: String!
    email: String!
  }

  type Query {
    users: [User]
  }

  type Mutation {
    addUser(name: String!, email: String!): User
  }
`;

// Mock Data
let users = [{ id: 1, name: "Alice", email: "alice@test.com" }];

// Resolvers
const resolvers = {
```

```
  Query: {
    users: () => users,
  },
  Mutation: {
    addUser: (_, { name, email }) => {
      const user = { id: users.length + 1, name, email };
      users.push(user);
      return user;
    },
  },
};

// Start Server
const server = new ApolloServer({ typeDefs, resolvers });
server.listen().then(({ url }) => console.log(`Server ready at ${url}`));
```

---

4. GraphQL with Express

```
npm install express express-graphql graphql
```

```
const express = require("express");
const { graphqlHTTP } = require("express-graphql");
const { buildSchema } = require("graphql");

// Schema
const schema = buildSchema(`
  type Query {
    hello: String
  }
`);

// Resolver
const root = { hello: () => "Hello GraphQL!" };

// App
const app = express();
app.use("/graphql", graphqlHTTP({ schema, rootValue: root, graphiql: true }));
app.listen(4000, () => console.log("GraphQL running on
http://localhost:4000/graphql"));
```

---

5. Advanced Concepts

- **Fragments** → Reusable query parts.
- **Subscriptions** → Real-time updates via WebSockets.

- **Batching & Caching** → Tools like DataLoader.
- **Federation** → Combine multiple GraphQL services.

---

6. Interview Tips

- **Q:** GraphQL vs REST?
  A: REST has multiple endpoints & can over-fetch/under-fetch. GraphQL has one endpoint, client specifies exact data, strongly typed schema.
- **Q:** How do subscriptions work in GraphQL?
  A: They use WebSockets (e.g., Apollo Subscriptions) for real-time updates like chat or notifications.
- **Q:** Challenges of GraphQL?
  A: Complexity in caching, over-complicated queries by clients, N+1 problem (solved via DataLoader).

☑ Next up is **19. Serverless with Node.js**.
Do you want me to cover **AWS Lambda, Azure Functions, Google Cloud Functions** with Node.js examples?