- Explain shortly about GraphQL, its purpose and some of its use cases
    - GraphQL is a syntax that describes how to ask for data, and is generally used to load data from a server to a client.

- Explain some of the Server Architectures that can be implemented with a GraphQL backend
    - Its like a personal assistant, you can ask for multiple things at once, and not make multiple calls to get what you want.

- What is meant by the terms over- and under-fetching in relation to REST
    - Over-fetching is getting too much information from a request, and under-fetching is not getting enough so you have to make more calls.

- Explain shortly about GraphQL's type system and some of the benefits we get from this
    - Graphqls type system, is a query where you can declare a new query with exactly what you need to request.
    - eliminates over and under-fetching
- Explain shortly about GraphQL Schema Definition Language, and provide a number of examples of schemas you have defined.
    - The graphql SDL is defined as a type and fields of that type.
        - type Post {
          id: String!
          title: String!
          publishedAt: DateTime!
          likes: Int! @default(value: 0)
          blog: Blog @relation(name: "Posts")
          }
        - type Blog {
          id: String!
          name: String!
          description: String
          posts: [Post!]! @relation(name: "Posts")
          }

- Provide a number of examples demonstrating data fetching with GraphQL. You should provide examples both running in a Sandbox/playground and examples executed in an Apollo Client
    - Data fetching ex1 (basic raw)

      ```
      {
        allPersons {
          name
        }
      }
      ```

    - Data fetching ex2 (Apollo client)

      ```
      import rp from 'request-promise';

      const resolverMap = {
        Query: {
          gitHubRepository(root, args, context) {
            return rp({ uri: `https://api.github.com/repos/${args.name}` });
          }
        }
      }
      ```

- Provide a number of examples demonstrating; creating, updating and deleting with Mutations. You should provide examples both running in a Sandbox/playground and examples executed in an Apollo Client.
    - Delete
      delete {
      delete-name(input: {id: xx)
      }
    - Update
      update {
      update-name(input: {id: xx, name: new name)
      }
    - Creating
      subscription {
        newPerson {
          name
          age
        }
      }

    - Mutation
      mutation {
        createPerson(name: "Bob", age: 36) {
          id
          name
          age
        }
      }
- Explain the Concept of a Resolver function, and provide a number of simple examples of resolvers you have implemented in a GraphQL Server.
    - A resolver is a function that resolves a value for a type or field in a schema. Resolvers can return objects or scalars like Strings, Numbers, Booleans, etc. If an Object is returned, execution continues to the next child field. If a scalar is returned (typically at a leaf node), execution completes

- Explain the benefits we get from using a library like Apollo-client, compared to using the plain fetch-API
    - Apollo platform contains a JavaScript GraphQL server that defines the schema and a set of resolvers that implement each part of that schema. The server is extensible in the sense that the commercial plugins can connect to any requests sent to the server.
    - Apollo provides a whole lot of open-source libraries that are extremely helpful in implementing GraphQL for JavaScript applications.

In an Apollo-based React Component, demonstrate how to perform GraphQL Queries, including:

- Explain the purpose of ApolloClient and the ApolloProvider component
    - Apollo Client is a complete state management library for JavaScript apps. Simply write a GraphQL query, and Apollo Client will take care of requesting and caching your data, as well as updating your UI.
    - ApolloProvider makes apollo client available to all components

- Explain the purpose of the gql-function (imported from graphql-tag)
    - GQL is a SQL-like language for retrieving entities and keys. The syntax for GQL queries is similar to that of SQL.

- Explain Custom Hooks used by your Client Code.
    - Custom Hooks are a mechanism to reuse stateful logic (such as setting up a subscription and remembering the current value), but every time you use a custom Hook, all state and effects inside of it are fully isolated.

- Explain and demonstrate the caching features built into Apollo Client
    -

In an Apollo-based React Component, demonstrate how to perform GraphQL Mutations?

- Demonstrate and highlight important parts of a “complete” GraphQL-app using Express and MongoDB on the server-side, and Apollo-Client on the client.
    -