GraphQl:
Periode 3 – Læringsmål:

- Explain shortly about GraphQL, its purpose and some of its use cases

GraphQL kan beskrives som et query language, som gør man kan tilgå API'er og data, på en lidt mere personaliseret måde. Med GraphQL behøver man hellere ikke flere endpoints, da man blot skal lave en query string til et enkelt endpoint, der indikerer hvad man skal bruge af data.

Ligheder mellem REST og GraphQL, er at de begge bruges til at lave Api'er, og kan bruges over http.

```
query
{
        books {
          id
          title
          author
          isbn
          price
        }
     }
```

- Explain some of the Server Architectures that can be implemented with a GraphQL backend

GraphQL Server can be deployed by using any of the three methods listed below −

- GraphQL server with connected database
- GraphQL server that integrates existing systems
- Hybrid approach

- What is meant by the terms over- and under-fetching in GraphQL, compared to REST

Men over-fetching fetcher vi for meget data, end vi skal bruge, dvs. der er data i vores response vi ikke skal bruge.

Under-fetching er omvendt, og det betyder, at vi er tvunget til at kalde flere endpoints, og muligvis benytte os af flere CRUD operationer.

Begge er performance issues, fordi enten bruger du mere bandwidth end du bør, eller også laver du flere http requests end du har brug for. REST returnerner jo prædefineret data struktur, der ikke er tilpasset.

Her vil jeg kun have bogens titel og pris:

```
query
{
      books {
        title
        price
      }
     }
```

Explain shortly about GraphQL's type system and some of the benefits we get from this:

GraphQLs query language er basalt set et spørgsmål om at vælge fields eller felter på et objekt.

F. eks

1. We start with a special "root" object
2. We select the `hero` field on that
3. For the object returned by `hero`, we select the `name` and `appearsIn` fields

Because the shape of a GraphQL query closely matches the result, you can predict what the query will return without knowing that much about the server. But it's useful to have an exact description of the data we can ask for - what fields can we select? What kinds of objects might they return? What fields are available on those sub-objects? That's where the schema comes in.

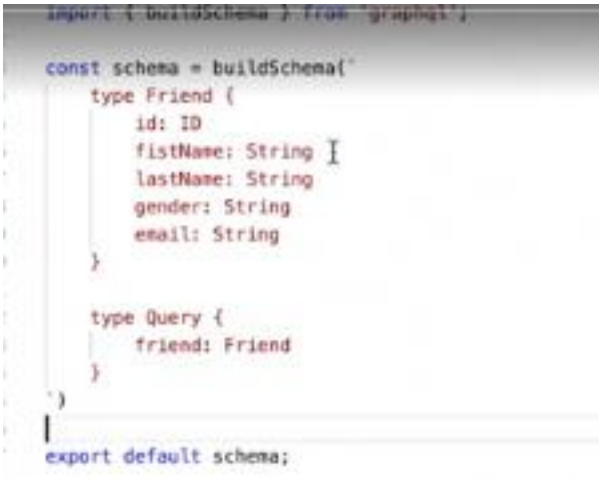
Every GraphQL service defines a set of types which completely describe the set of possible data you can query on that service. Then, when queries come in, they are validated and executed against that schema.

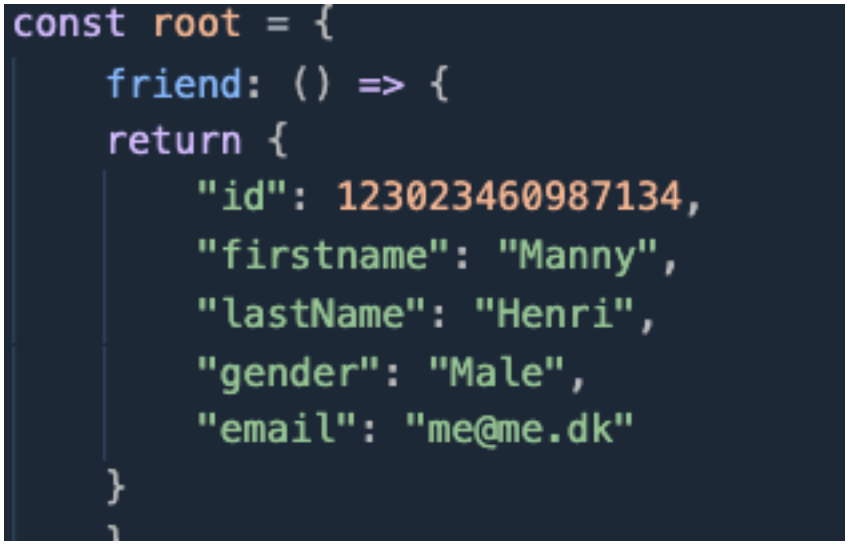
- Explain the Concept of a Resolver function, and provide a number of simple examples of resolvers you have implemented in a GraphQL Server.

Schema definerer query type, resolver henter data. Type definition what data we expect, resolver gets it.

Kig I schema.js filen og se at den tager en type ”friend” og vi definerer hvad en friend tager: tager et iD, firstname, email, gender osv.

Vi skal også have defineret en query type, som er en friend. Så hvis jeg spørger efter en friend skal jeg retuerner en friend type.





Resolveren, bemærk den følger samme schema/felter.

- Mutations:

Er Graphql måde at lave crud operationer.

. A query does not change the state of the data and simply returns a result. A mutation on the other hand, modifies the server-side data. So, for CRUD operations, you would use a query to *read* and a mutation to *create*, *update*, or *delete*. The following table shows how CRUD relates to GraphQL.

CRUD	**SQL**	**GraphQL**
Create	INSERT	MUTATION
Read	SELECT	QUERY
Update	UPDATE	MUTATION
Delete	DELETE	MUTATION

1.NPM init
2. `npm install --save-dev @babel/preset-stage-0`

3. npm install express express-graphql graphql nodemon (nodemon, giver også muligheden for at gemme og ændre koden, og starter serveren om autoamtisk).

4. lav index.js og .babelrc.

5. gå til package.json filen og linje 7 og skriv scriptet til at starte serveren , (nodemon og vi starter med index filen og executer med babel node.

```
"start": "nodemon ./index.js --exec babel-node -e js"
```

6. quick preset I .babelrec filen, som vi har installeret (envoirnment og stage-0):

```
{
            "presets": [
                "env",
                "stage-0"
            ]
        }
```

7. I index.js creater vi main server.

Import express from ”express”

Og lav en variabel const app =express() , hvor vi giver den serveren

App.get(’/’) get endpoint (request og response)

Base server er klar, nu skal vi have graphql på:

8. vi laver et schema (schema.js).