TP-10

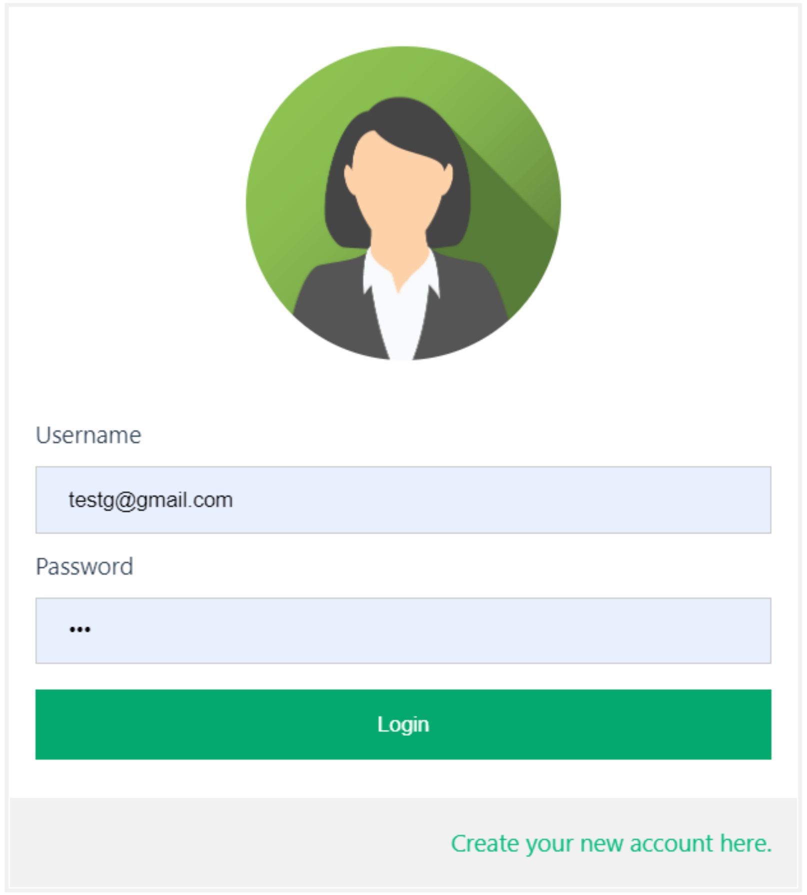
VueJS, NodeJS

Authentication (conti.)

TP10 Exercise

TP10.1: Frontend auth

In your VueJs app, integrate the previous authentication APIs with VueJS



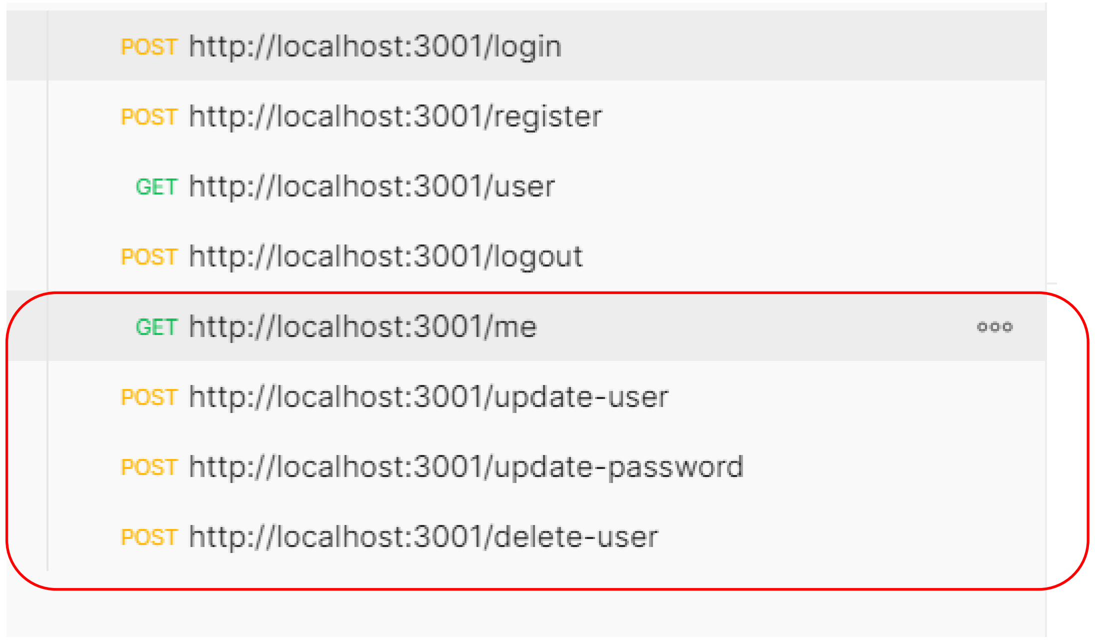
Make sure:

- The token is stored in the browser cookie after login
- The home page is reachable unless the token is existed
- The token can be removed by just calling the logout API or coming to the expired date.

TP10 Exercise

TP010.2: More auth Apis

In your nodejs backend, continue implementing the following authentication and user APIs

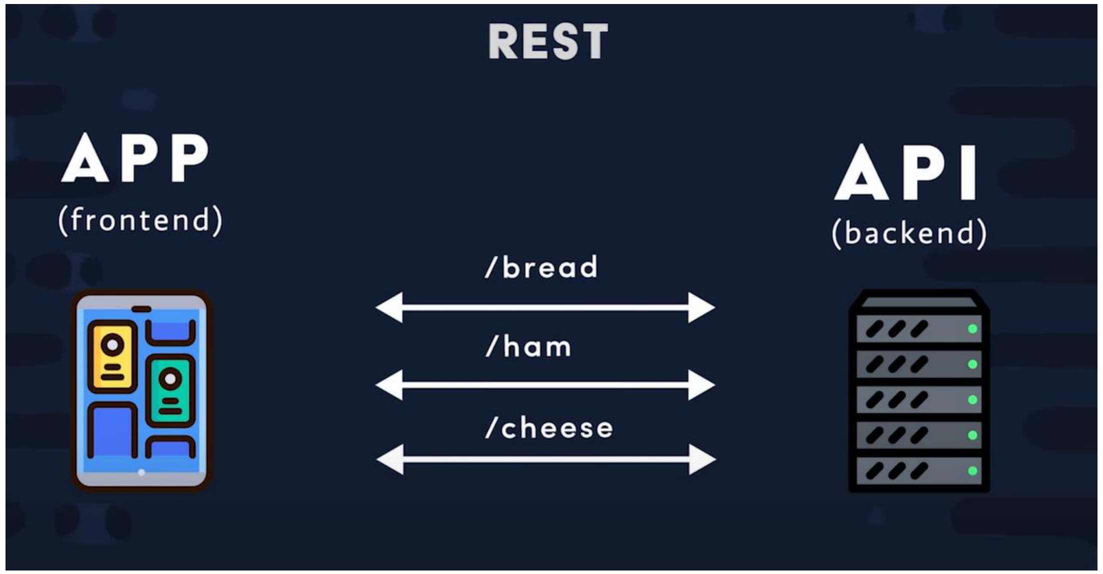


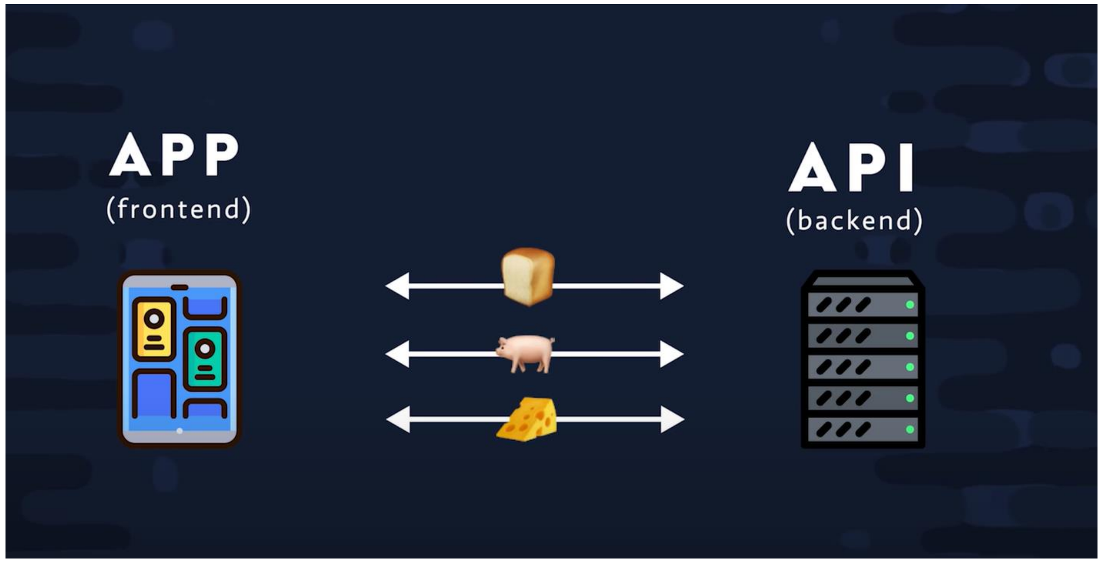
POST http://localhost:3001/login
POST http://localhost:3001/register
GET http://localhost:3001/user
POST http://localhost:3001/logout
GET http://localhost:3001/me
POST http://localhost:3001/update-user
POST http://localhost:3001/update-password
POST http://localhost:3001/delete-user

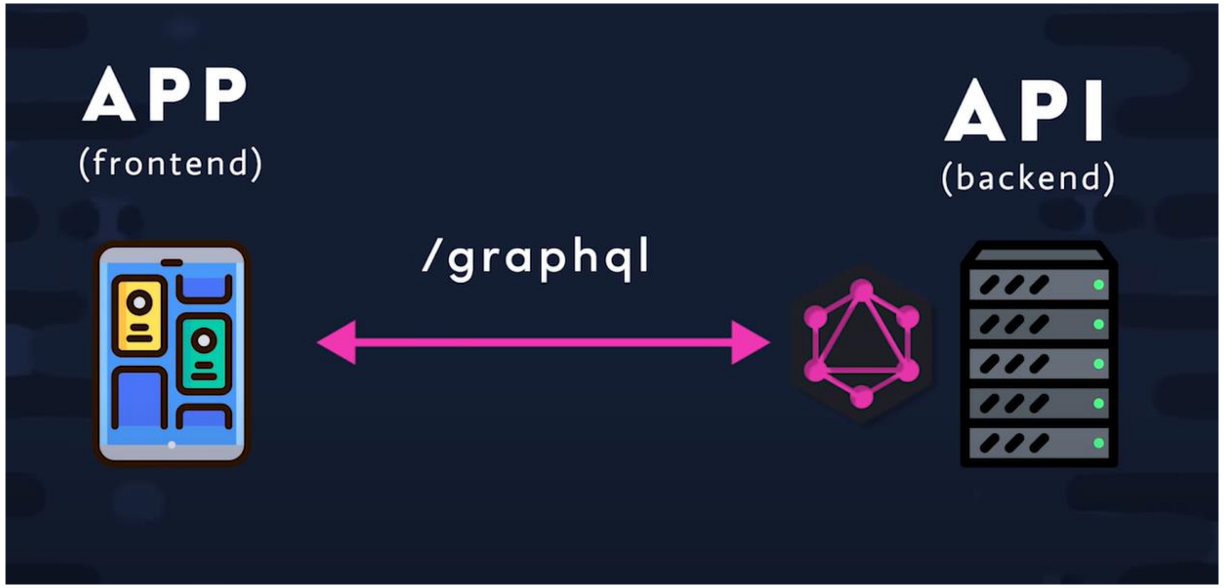
Getting to learn another
new Thing

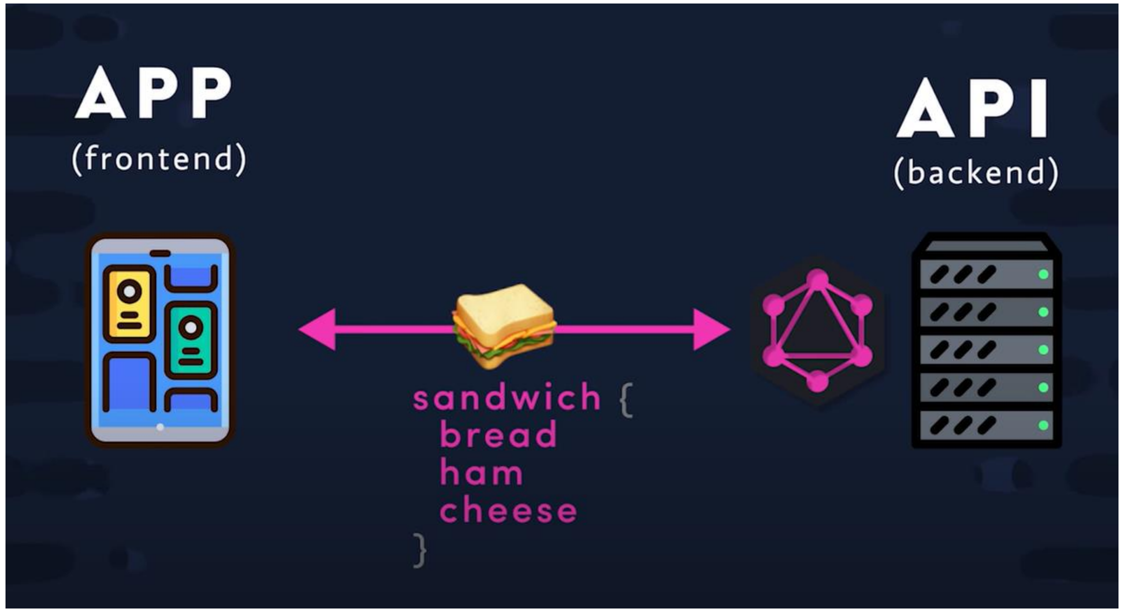
“GraphQL”

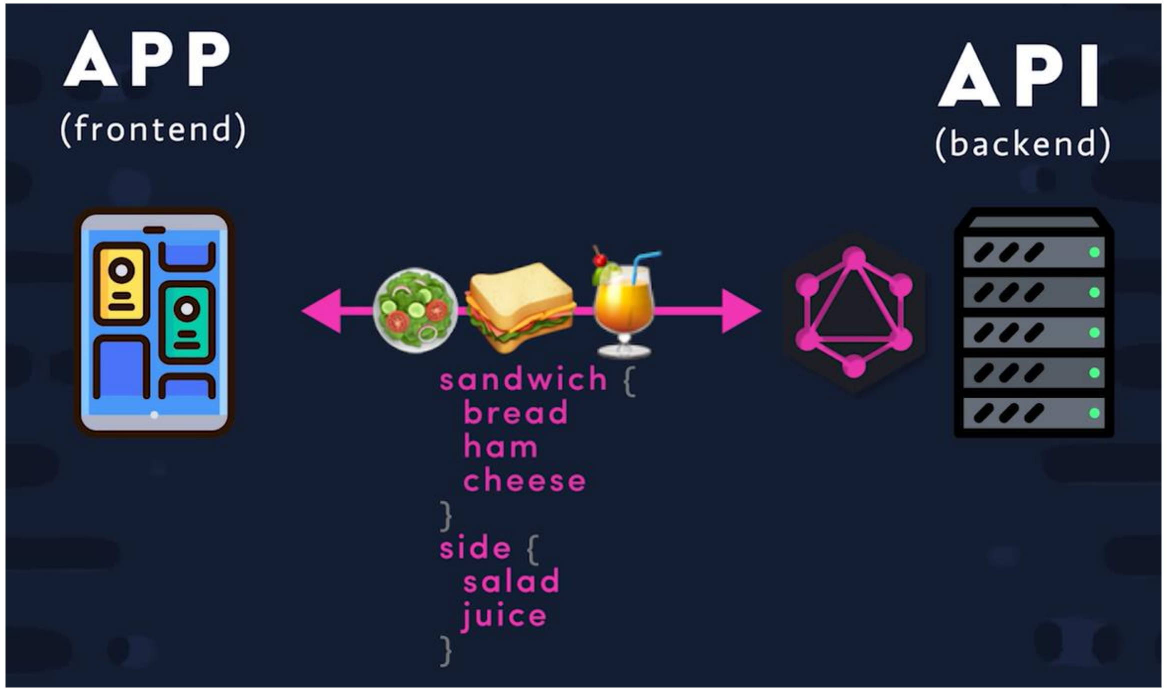
REST vs GRAPHQL


REST
APP
(frontend)
/bread
/ham
/cheese
API
(backend)


APP
(frontend)
API
(backend)


APP
(frontend)
/graphql
API
(backend)


APP
(frontend)
API
(backend)
sandwich {
bread
ham
cheese
}



GraphQL playload with spaceX APIs🚀🚀🚀

https://api.spacex.land/graphql/

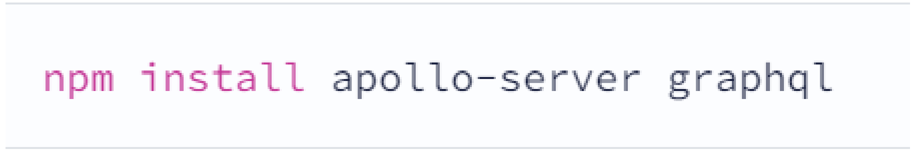
Get started with a simple installation

https://graphql.org/learn/

https://www.apollographql.com/docs/apollo-server/

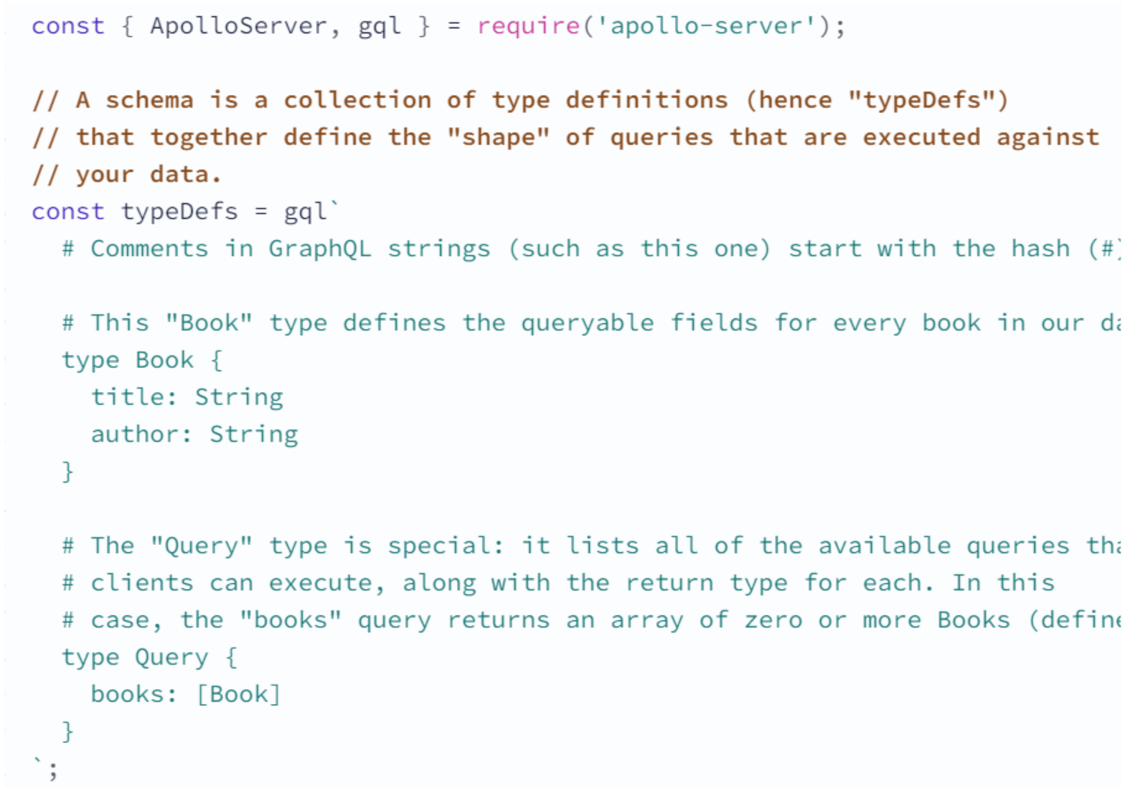
+ Apollo Server

1. Install dependencies



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

2. Define your GraphQL schema



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

// A schema is a collection of type definitions (hence "typeDefs")
// that together define the "shape" of queries that are executed against
// your data.
const typeDefs = gql`
  # Comments in GraphQL strings (such as this one) start with the hash (#

  # This "Book" type defines the queryable fields for every book in our d
  type Book {
    title: String
    author: String
  }

  # The "Query" type is special: it lists all of the available queries th
  # clients can execute, along with the return type for each. In this
  # case, the "books" query returns an array of zero or more Books (define
  type Query {
    books: [Book]
  }
`;
```

3. Define your data set

```javascript
const books = [
  {
    title: 'The Awakening',
    author: 'Kate Chopin',
  },
  {
    title: 'City of Glass',
    author: 'Paul Auster',
  },
];
```

4. Define a resolver

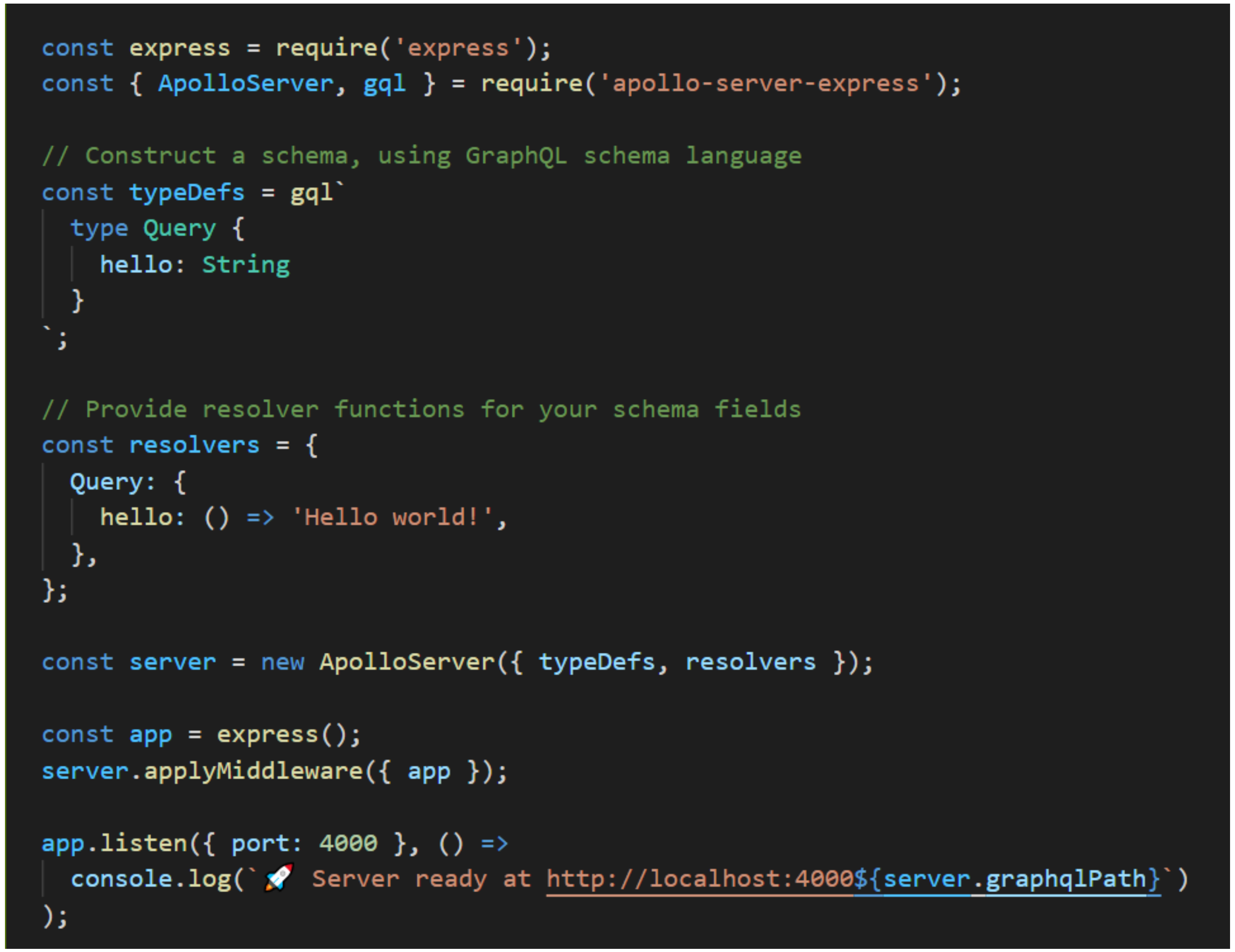
```javascript
// Resolvers define the technique for fetching the types defined in the
// schema. This resolver retrieves books from the "books" array above.
const resolvers = {
  Query: {
    books: () => books,
  },
};
```

5. Create an instance of ApolloServer

```javascript
// The ApolloServer constructor requires two parameters: your schema
// definition and your set of resolvers.
const server = new ApolloServer({ typeDefs, resolvers });

// The `listen` method launches a web server.
server.listen().then(({ url }) => {
  console.log(`🚀  Server ready at ${url}`);
});
```

To sum up:



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

// Construct a schema, using GraphQL schema language
const typeDefs = gql`
  type Query {
    hello: String
  }
`;

// Provide resolver functions for your schema fields
const resolvers = {
  Query: {
    hello: () => 'Hello world!',
  },
};

const server = new ApolloServer({ typeDefs, resolvers });

const app = express();
server.applyMiddleware({ app });

app.listen({ port: 4000 }, () =>
  console.log(`🚀 Server ready at http://localhost:4000${server.graphqlPath}`)
);
```

Your GraphQL API should be running at http://localhost:4000/graphql 😋

Good luck 👌