Part 8 & Part 9

Introduction

Self Evaluation: Mastery / Progressing / Needs Improvement

Due date: 2021-10-29

Completion: 2021-10-29

GitHub: https://github.com/ZDGharst/FullStackOpen

Part 8 – GraphQL

Part8a teaches the student about GraphQL: specifically, creating a GraphQL schema on the backend. GraphQL, developed by Facebook, is a popular alternative from an API developed with a REST architecture. The request forms a query describing the data wanted, and the API replies with the data in turn. Within this section, I play with the GraphQL playground, create a GraphQL backend using Apollo, handle errors, use mutations to modify data, and create context for authorization purposes.

The next part is React mixed with GraphQL. *Par8b* has me use Apollo client on the frontend to make queries that parse and store the data within the cache; mutations are performed which change the data on the backend but also the cache on the frontend. Lastly, I learn how to save the application's state to the local cache – no Redux needed.

Within *Part8c* I have to use MongoDB to store the data rather than storing data in memory. This means creating a MongoDB schema, storing the data, and retrieving/populating it when the GraphQL requests it from the MongoDB. Validation is also down on the MongoDB end, which is forwarded to the requester of the GraphQL client.

Part8d is about now incorporating the changes to the backend in *Part8c* to the frontend. Now, a user account is required. A login query is made to the GraphQL API, and a token is replied on success. The token is stored in the application state. Every GraphQL query sends the token in a header (configured by the programmer) which is checked against the backend to see if that user is authorized to make the query it's trying to make.

The last part, *Part8e*, teaches how to create fragments which can be reused to avoid code duplication. I also learned how to create subscriptions both serverside and clientside (although, the course is written for Apollo 2.0 and the latest version, 3.0, completely changes subscriptions). I also cover the n+1 problem – that a query to a database can cause n more queries, such as viewing a friend's list of a single person where n is the number of friends.

Additional notes can be found in the Part 8 README.

Part 9 – TypeScript

I was very excited for this part about TypeScript. *Part9a* is mostly a precursor about TypeScript. TypeScript is an open source programming language created by Microsoft. It's a superset of JavaScript, as in, it has all the features of JavaScript plus more. TypeScript is compiled to JavaScript in order to be ran, but TypeScript allows for static code analysis which is extremely important when working with big, enterprise applications. It also allows for structural typing, type inferencing, and type annotations. It's more difficult to forget to cover for edge cases when programming in TypeScript compared to JavaScript, and the overhead is non-existent on production code as it's compiled down to plain JavaScript.

Creating a TypeScript project and getting started is the purpose of *Part9b*. A lot of packages are needed for start up in order to add much needed types to get started. Then, I set up my tsconfig and linter to enforce certain TypeScript rules that are optional. I create my own types, and use them within the programs created by the exercises. I create an API using Express with TypeScript within this part as well.

The Express part previously created still needs a bit more work on creating types and interfaces in *Part9c*. A database of patients is created from a flat json file; this is parsed per field and typed appropriately. There are also utility types for the parsing steps. One of the best parts about TypeScript is here: the linter warns that when searching for a match in an array, there's a possibility of the returned result being null, and it will require you to handle that case.

Lastly, *Part9d* deeals with TypeScript in the frontend for React apps. This is a very heavy section, however, it's mostly covering the same material done previously for Part 1 and Part 2 but with TypeScript specific annotations. The state is handled internally with a Flux-like architecture without using any built-in librarys such as flux. A new part of React hooks is covered here: useContext and useReducer. Overall, this part of the course had a lot of exercises, and I probably spent twice as long on Part9 as I did any other part. There are still a few exercises left to complete that I will be continuing into my next unit.

Additional notes can be found in the Part 9 README.

Deliverables

For a completed list of exercises, please view the commits on the master branch here. There is a history browser, diff viewer, and the git messages explicitly define exercises.

All exercises except 9.22-9.27 were completed in this section. Part9 was very intense, and I will be finishing these in the next module which is a bit lighter.

Supplementary Material

Modularizing your GraphQL schema code

Thoughts on Structuring your Apollo Queries & Mutations

React Hooks Tutorial