Report 1

Viktor Enghed

February 11, 2021

The first step was to choose a strategy and a framework. We wanted to create a client-rendered application with a REST backend. We chose to use React as the frontend as it is extremely prevalent on the market today and has great documentation. For the backend we chose node and express.js for the same reasons as well as wanting to learn more about it.
At first we wanted to use MongoDB as the database, mostly because it was easy to implement and made sense to us, but we had problem motivating why to use a document database and after testing out Sequelize with a PostgreSQL database hosted on Google Cloud Platform we chose to not go with MongoDB.
After familiarizing ourselves with the chosen frameworks and their node implementation we chose to structure our backend using the MVC + integration pattern, taking inspiration from the provided object-oriented example on the course website. We chose not to use classes but instead to use a more functional approach using ES6 modules and group code into layers in that way instead. At the moment, the controller layer is just passing on function calls from the HTTP(view) layer to the integration layer so it might be removed.
We created several documents and virtual whiteboards we could use to plan out our work and to keep track of our architectural decisions. We chose to use GitHub for version control and Heroku for hosting our application. This was very quick and easy, but can be modified to run tests before deploying. How it is set up now, each push to the main branch in our repository on GitHub will trigger a deploy on heroku. The create-react-app bootstrap comes with a nodemon-like development server which is very convenient to use, but requires some setup so that we proxy all our requests to the backend server we are also developing. When pushed to Heroku, the React app is built into an optimized production build and is served by the backend server.
We tried to make sure that the code conventions are consistent through both the backend and the frontend. There are several ways to handle asynchronous code for example. We chose to use promises instead of async/await as we think it makes the code more readable. async/await has some optimization capabilities but as far as we can tell, these optimizations are not live in V8(node and Chrome), SpiderMonkey(Firefox) or Chakra(Edge). Furthermore 15 000 visitors in two weeks is a small amount of traffic so performance should not be an issue.