Architectural Decisions

ALEX LINDBERG alex5@kth.se
DENNIS HADZIALIC denhadh@kth.se
LUCAS LARSSON lulars@kth.se

March 10, 2023

1 Introduction

This document contains our notes on all architectural decisions made during the development of our job-applicant application, in order to fulfill requirement #30 from the *tasks-affecting-grade.pdf*. We divide most of the topics discussed by their position in the stack, i.e. backend, database or frontend.

Sub-section titles may contain a hash with a number, e.g. #16, which corresponds to the task number in *tasks-affecting-grade.pdf*

2 Backend

2.1 Package Manager

For the backend we do not need a build-tool to set up our server, but obviously we use some kind of package manager, in this case `npm`. Npm is a great tool when working with a `node.js` backend since it makes managing dependencies super easy.

2.2 Framework

For the backend we chose Node-express. This because the application does not require much computational blocking code. This is because it's more data-heavy with requests rather than processing data. All the developers are already familiar with working with JavaScript so it's a great choice.

2.3 OpenAPI/Swagger

We chose to document all the api's using swagger docs. This is because it's standardized and with express-openapi-module you can validate all the request data automaticlly with the specified swagger docs. This leads to less time to make sure all the in-data is correctly sent to the server with correct types/format. This also leads to future developer to easily read what the api's are designed to do, output and input.

2.4 Error handling

For the errors we wanted to standardize them with custom Boom-objects which are used to give informative errors depending on what has gone wrong.

2.5 ACL

We created an ACL which is made to authenticate if the user is allowed to access/change the the requested resourse. For example an applicant can't view other applicants applications but a recruiter can se all. We chose RBAC for this because we though that role-based was the superior choice.

2.6 Transactions

We chose to implement transactions in specificlly when posting a new application to the server. This because when creating an application it is vital that the data is stored in both tables. We chose not to do so when signing up a new user. This because when you signup you will have a new user created and a session. But it is not vital to have the session created. Worst-case user have to login which is not a deal-breaker.

3 Database

Our choice of database for this project is Postgres. Postgres is a well supported, well maintained and easy to use relational database that everyone in the team feel comfortable working with. Not much discussion was had around options except for whether or not to use any other type of database e.g. a document database like MongoDB. We believe there was no need to explore that option for this project since we won't be keeping and/or querying large datasets and can easily just build off of the database provided in the project.

3.1 Dependencies

For password encryption we use a postgres package called `pgcrypto` which performs hashing and salting for us as well as making encryption and decryption quick and easy.

4 Frontend

4.1 Build Tools

This section makes a (sort of) long discussion short.

We decided to initialize the product using **Vite** instead of the trusty **Create React App** for this project. **CRA** was the preferred method for a long time to setup a new React frontend, but with a number of years behind it it's started to show. Problems you can run into with CRA is for instance if you ever need to change anything surrounding the Webpack configuration (which CRA uses as a bundler). You would either need to solve this by running **eject** which can be a very bad experience since everything CRA is doing for the Devs is now put entirely on the Devs to manage themselves, and you **cannot** go back.

Unlike CRA, Vite uses ESBuild for transpilation and minification which is a faster (because ESBuild is written in GO-lang) and Rollup for bundling. To add to that, Vite supports hot module reloading (HMR) which can greatly speedup development load times. More information on it's strengths and weaknesses can be found here.

The final reason we went with Vite was related to the following section on TailwindCSS. Tailwind works with CRA, however you may need to apply some hack fixes to get it to work since Tailwind is not supported by the Webpack version CRA uses, therefore requiring a fix such as CRACO.

4.2 TailwindCSS

For this project we decided to go with TailwindCSS to speed up the styling frontend styling process and to reduce the amount of files cluttering the repository. Tailwind also decreases the clients bundle size in production thanks to the purging of unnecessary and redundant CSS directives.

4.3 State Handling

We made the decision to not use a framework like Redux to handle state simply because we believe the app will be small enough to not warrant it. If, however, it turns out we need one later we want to go with a lighter framework, perhaps Zustand if we need a state machine, otherwise maybe Jotai if we only need globals.

React-query that we use for querying the backend also handles caching, so we should be fine if we keep to our design of loading in the list of applicants and maybe use a modal for applicant details when we need it.

5 Deploying to Production

This section is divided into three sections in the same way they are deployed and hosted.

5.1 frontend

The frontend application is built using the **Vite** build tool, and after doing some research we found a deploying service **Vercel** that is built for quick, scalable deployment. As such we choose it to deploy the frontend application.

5.2 backend

When considering the backend deployment, we wanted to deploy using the same service used to deploy the frontend application, for the sake of uniformity and having all deployments on the same provider.

A problem we faced then is that **Vercel** does not facilitate using stand alone Express application, it requires it to have server-less architecture. This is seen as a drawback since we can not use Docker for deployment in that case.

Docker is great for controlling dependencies and specific package version.

And giving the restriction/limitation in **Vercel**, this is considered a deal-breaker, we decided to change the provider, as we did not want to remove docker and change our backend architecture just for the sake of deploying on **Vercel**.

As a result we decided to use GCP (Google Cloud Platform) where we use GCP's Cloud Run service to deploy backend.

5.3 Database

The database is hosted on GCP's Cloud Storage as a manged database system as a postgresql instance.

6 Miscellaneous

6.1 Handover

6.1.1 Cloud Provider #18

The application is live on the cloud, as mentioned in section **6 Deploying to Production**.

Google Cloud Platform **GCP** for the backend (CloudRun) and (CloudSQL) for the database. **Vercel** for frontend.

6.1.2 Documentation #19

All source code is available on GitHub. We use a Mono-repo, which means both the frontend and backend are in the same repository, We have a Documentation directory, where all documentation is saved, including this document and other documents that are referenced here.

In addition to this Documentation folder, all reusable components are annotated with documentation in the source code. and the backend API's are documented using OpenAPI-Swagger.

7 Development environment

A brief description of the development process is as follows, we use git branches for features, after a feature is considered done by the developer, a pull request is sent against the development branch and deployed to the staging environment.

We have a complete replica of the whole app live as a development environment that we can test before releasing it to production.

Since we use the development database for the staging app, we do not want developers connecting to it for testing, so we use docker-compose which creates a locally isolated instance of the database.

8 Other

See the **deployment routines** document for additional information regarding deployment.