SMAT Project documentation

Alvaro Tedeschi
Mateus Nobre

V 1.0

October 5th 2021

Contents

Chapter 1

Application

1.1 Application Description

SMAT's purpose is to provide a unique place where users can easily share and search for academic documents while also being able to interact. It is a social network web application aimed at people with the desire of helping each other in their academic studies.

It features:

- Publishing academic documents.
- Social interacting through posts comments.
- Searching for published academic documents.

1.2 Requirements

SMAT uses a incremental model strategy in development and therefore its requirements will constantly change. However, there are key requirements that make up the base of the application, they can be listed as:

- Application should have user authentication built in.
- Registered users should be able to publish any document they like.
- Registered users should be able to like and make comments to any other user's publication.
- Registered users should be able to like other user's comments.
- Registered users should be able to follow other users.

- Anonymous and registered users should be able to easily search for any documents published.
- Anonymous and registered users should be able to search for registered users.
- Registered users should have a profile page.
- Registered users should be able to get the publications of the users they follow on their home page.

1.3 System Stack

SMAT uses the PERN stack :

- PostgreSQL: Object-Relational Database
- Express: Back-End Framework
- React: Front-End Library
- Node: JavaScript runtime environment

The stack also includes Google Cloud Services, as it provides a storage bucket to be used as database for larger files such as pictures or pdf files. Finally, in production, the system is deployed to Heroku.

1.4 System Architecture

1.4.1 Development Architecture

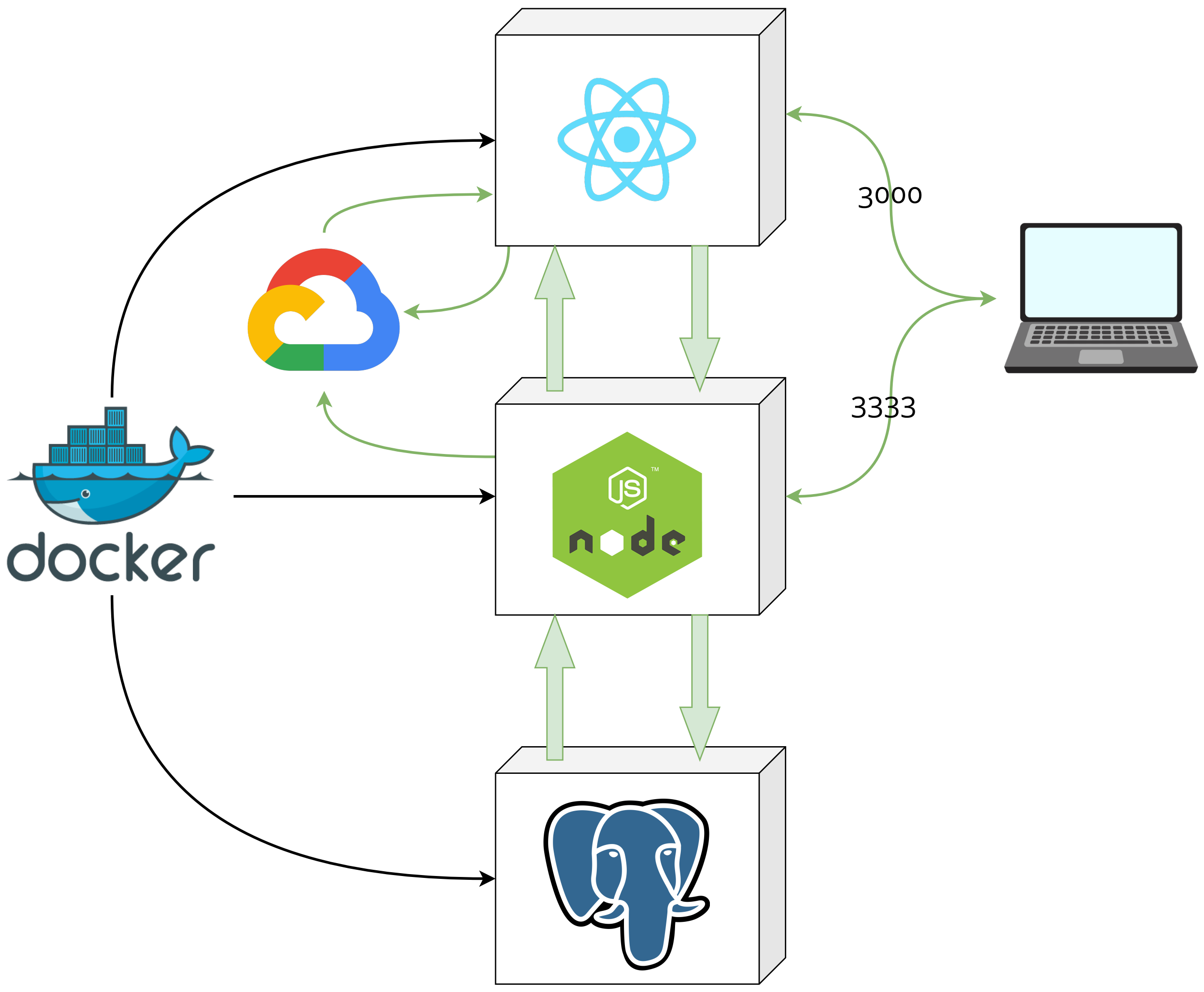

Figure 1.1: *SMAT* Development Architecture

The development architecture uses docker and docker-compose to orchestrate 3 containers for React, Node and PostgreSQL. The react container is set to be open on port 3000 and node on port 3333 on the user's local environment. The React container serves the front-end and makes fetch requests to Google Cloud storage bucket as well as the Node container containing the back-end. The Node container communicates with PostgreSQL's container to be able to storage and fetch relational data.

1.4.2 Production Architecture

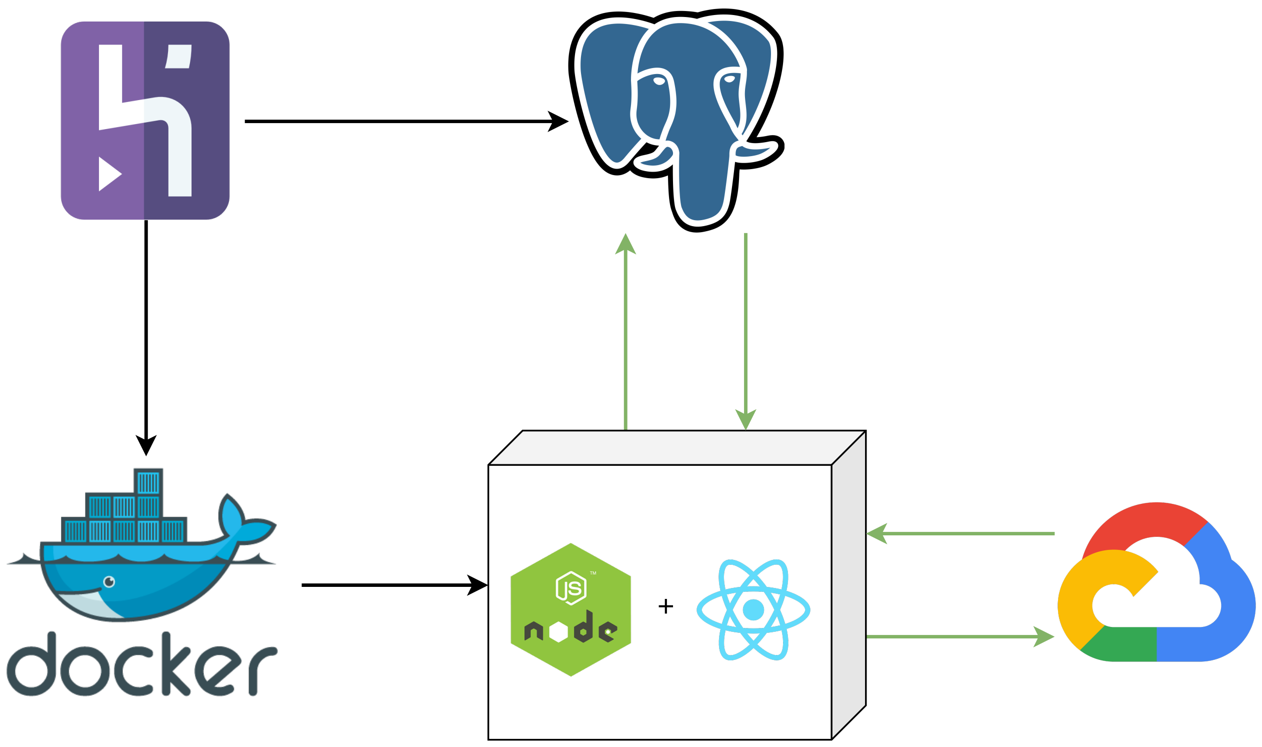

Figure 1.2: *SMAT* Production Architecture

The production architecture uses Heroku to manage the PostgreSQL database and to build and run the docker image containing a Node instance which will server side render the static files provided by the React build. This Node instance is also responsible for communicating with PostgreSQL and Google Cloud databases.

1.5 Relational Database Model

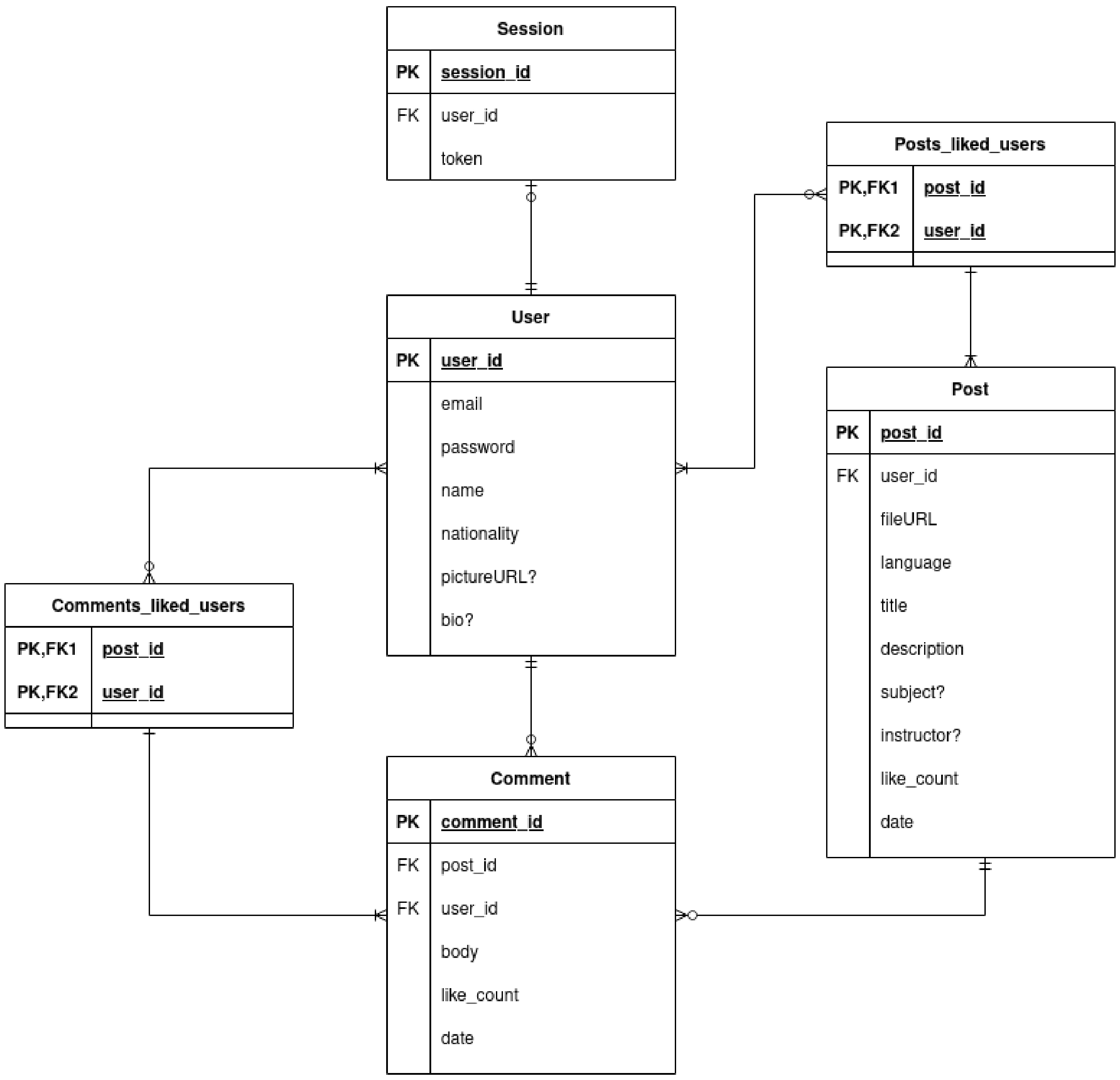

Figure 1.3: *SMAT* Relational Database Model

The database model entities are used as it follows:

- User entity holds user information.
- Session entity holds tokens used for authentication for each user.
- Post entity holds user's posts information.
- Comment entity holds users's comments information.

- Posts Liked Users holds posts like mappings to users.
- Comment Liked Users holds comments like mappings to users.

Chapter 2

Project

2.1 Dev Team

Team Member	Role
Alvaro Tedeschi	Back-end developer
Mateus Nobre	Front-end developer

Table 2.1: Dev Team

2.2 Modules and Packages

2.2.1 Typescript

Instead of using usual Javascript as the backend programming language the team chose to use Typescript which is a superset of Javascript that adds types to objects. This helps maintain code without over commenting functions specifying parameter types as they are built in by default and can be viewed directly from your code editor.

2.2.2 TypeORM

In order to take advantage of using Typescript and PostgreSQL together the team opted to use TypeORM as one of the backend dependencies. TypeORM is a ORM (Object-relational mapping), which in turn means it is able to map object-oriented programming to post-greSQL commands. Using TypeORM with Typescript provides a very good developer experience as it makes maintaing and understanding database queries and entities definitions very easy.

2.3 Implementation Steps

The project was separated into steps in order to organize its development process:

1. Initial development setup with Node, Express and React (Done)
2. Containerize application using docker-compose (Done)
3. Connect containers and services in development (Done)
4. First build and deploy (Done)
5. Create initial database model (Done)
6. Define entities and basic creating, updating and deleting functions (Done)
7. Back-end authentication handling (Done)
8. Add back-end routes for users, posts and comments (Done)
9. Connect and setup Google Cloud Storage (Done)
10. Finish v1 back-end routes and run migrations in cloud (Done)
11. Setup React authentication (Done)
12. Add simple components to React (Done)
13. Finish v1 front-end (Done)
14. Setup SSR (In progress)
15. Deploy v1

2.4 Challenges and Achievements

2.4.1 Docker

Learning to use docker and docker-compose to provide an equal development environment for all team members was definitely a hard task. However, using it was a great choice as it makes spinning the whole application really easy and fast.

2.4.2 Heroku Deploy

After getting everything working using docker-compose in development the team then needed to deploy our application to Heroku which unfortunately doesn't support docker-compose, this meant they wouldn't be able to deploy every service to a single domain. This problem was tackled by using Heroku's managed PostgreSQL database and serving the React application using the back-end, this way all was needed was to deploy the Node application in a single container.

2.4.3 Server-side rendering React

Currently SSR (Server-side rendering) React hasn't been fully achieved as production is only able to render React's app root route. This is being addressed and will be fully implemented in the future. For the time being, only the development environment supports rendering React to its full extent.

2.5 Current Project State and Future Plans

Currently the project only lacks SSR to be fully deployed to the cloud in its first working version (v1). Since SSR with pure React poses a threat to the current project structure the team has been looking into NextJs, a React framework designed to support SSR from the get go.