Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Puebla

TC2026

Web Development Laboratory
Final Project

Students:
Salvador Orozco Villalever - A07104218
Rafael Antonio Comonfort Viveros - A01324276
Date: May 9th 2018
Professor: Dr. Adolfo Centeno Téllez

Contents

1 Problem definition

For this project, we were tasked to extend the implementation of the system to satisfy the T-Crum project requirements that we had worked on during the 2nd partial. T-CRUM is a system oriented for students and professors, with the objective of helping them with the management of projects based on the Scrum methodology. This means that there is a need for managing sprints, user stories, tasks and other components common to agile methods. We were asked to develop the backend (database and REST API) and frontend (GUI and website). There was also the requirement to do the former with PostgreSQL/NodeJS and the latter with Angular 4. Moreover, MongoDB was to be used to store the non-relational data items of each project and other components of the system. Besides, MySQL and Laravel were to be used for the storage of the logs, i.e. the history of queries against the database. The use of different database managements systems is convenient to make the project more modular, flexible, secure and scalable. The use of more than one server to run the database management systems on adds to these layers of security and scalability.

2 Implementation

Explanation of implementation

For the back-end, we followed an identical structure to that found at the provided scotch.io tutorial. For the main application back-end, we used PostgreSQL as databases and Sequelize to manage the Rest API. First off, we define Sequelize models, that are the equivalent of tables but for the ORM. They are used as objects to hold the data retrieved from and sent to the database. Migration files are used to make a translation into Postgres instructions, which will make modifications to the target database. Seed files do as their name indicates, and perform tuple insertions to populate the database. The Middlewares used are exclusively to allow authentication to users through tokens, forbidding any request, which doesn't include an authorization header with a JWT token generated by the application, from accessing the REST API. The JWT service file is used by this middleware. Speaking of which, a controller of a particular model is in charge of its CRUD, an Authorization controller generates JWT tokens with a ten minutes duration for properly identified users. Functions that exploit the methods of a Sequelize Model are defined to work with the request make to the API: create, update, list, retrieve, delete and other elaborations of the former were added to some controllers. The index file at the routes folder, binds each one of this methods to a path in the URL, along with a request method type (get, put, post, delete). Outside of the server directory, in

bin/www, the back-end server is ran by the NodeJS library Express.

Meanwhile, on the front-end, we also followed the Angular standard by exploiting its components layout. A component defines, through TypeScript, HTML and CSS files, a view to perform an action in particular. In our case, we wanted to make a component for each action of the CRUD. This way, we would be able to integrate several screens embedded in one another. For example, the sprint's retrieve component uses the user story list component, which is independent and can be utilized on its own. This means that along with the sprints fields, you can see all the user stories associated to it. Similar as to the back-end, we use models to declare an object which is used to transfer data, analogous to the DB counterpart. Guards guarantee that all users are logged in to do anything in the system, furthermore the token provided to the session is included in every request made to the backend. Finally, the services provide extra functionality to the components, a special service guarantees that a log is saved for each create, update or delete operation. The CRUD service provides a generic way to match each component to its REST API call, in one generalized place. It also had the side effect of simplifying the functions in a component's TypeScript, where the requests are made, and their responses analyzed. An special service to handle the three backend error responses was included, it also displays messages properly.

Public repository link:

Main repository

3 Architecture

The architecture is composed of three docker containers for the following elements:

1. The backend: based in NodeJS and the Sequelize ORM
2. The frontend: based in Angular, using JS, TS, HTML and CSS
3. The database: an RDBMS for PostgreSQL

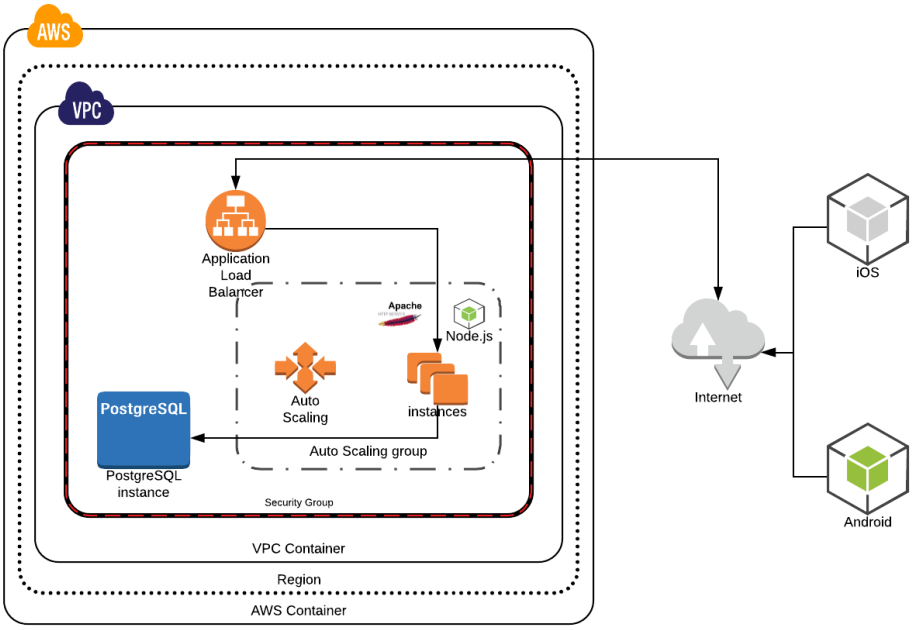

Figure 1: Application Architecture

4 Conclusions

This project allowed us to develop and architecture a robust web application using several technologies seen in class, such as Angular, Node, Sequelize, PostgreSQL, Docker and Jenkins. It was indeed a difficult project and we did struggle. Nevertheless, we learned a lot from it and we're looking forward to developing other projects with similar schemas regarding continuous integration, testing and a modular architecture using containers and the latest trends in virtualization.