COMP3900-H15A-capSquad - Project Report

Daniel Latimer z5115175
Connor O'Shea z5115177
Kevin Chan z5113136
Oliver Richards z5157383

Peter Kerr
z5115807

November 11, 2020

Contents

1 Overview **2**
1.1 System Architecture 2

2 Functionalities and Implementation Challenges **2**
2.1 Functionalities 2
2.2 Implementation Challenges 2
2.2.1 Backend Stack 2

3 User Manual **3**
3.1 Software Setup Instructions 3
3.2 Site Usage and Functionality Guide 3

1 Overview

1.1 System Architecture

2 Functionalities and Implementation Challenges

2.1 Functionalities

2.2 Implementation Challenges

2.2.1 Backend Stack

The backend of UltraCast employs an unusual technology stack, with MongoDB as a persistence layer, flask as a webserver framework and graphql (via graphene and graphene-mongo libraries) as an API layer. This created difficulties in implementing common web-app functionalities due to (1) a lack of documentation on the libraries being used and (2) no online examples implementing these functionalities with this stack.

User Authentication Implementing user authentication for the backend was a non-trivial task because the Graphene and Graphene-Mongo libraries which are used for the API layer do not natively support this functionality. A major challenge in applying general purpose authentication libraries, for example flask-jwt[1], is that only one route is used for all API calls. Some of these API calls need to be authenticated e.g. deleting a podcast where others should not be e.g. signing up to the site. The Flask-GraphQL-Auth library[2] provides the required authentication methods, however, it is not actively maintained. After much research, user authentication was implemented using the flask-jwt-extended library[3]. This library allows authentication to be required on a per-function level, rather than for an entire route. Hence, certain mutations and queries can be protected with user authentication where required. The frontend calls a signin mutation which returns a Json Web Token (JWT). This mutation does not require authentication. The frontend then stores this JWT as a cookie and sends it in the header of any future GraphQL API requests.

Resolving Nested Queries While testing the frontend, it was discovered that some backend GraphQL queries were taking upwards of one minute to return. The site was still responsive, however it took a long time for recommended podcasts to be displayed. Further investigation revealed that where nested references were used in the database models, and the GraphQL query involved dereferencing these references, the Graphene-Mongo library would perform one database operation per parent node. These database operations are performed sequentially. Since the MongoDB instance is hosted in the cloud, each database operation takes some number of milliseconds due to network latency. When a large number of parent nodes were fetched, this resulted in very slow queries. It was not feasible to modify the Graphene-Mongo libary to issue less database operations. Hence, the decision was made to move the GraphQL API webserver to the same cloud container as the MongoDB instance. This improved the time for some queries from over fourty seconds to less than a second.

[1] Available at https://github.com/mattupstate/flask-jwt
[2] Available at https://github.com/NovemberOscar/Flask-GraphQL-Auth
[3] Available at https://github.com/vimalloc/flask-jwt-extended

3 User Manual

3.1 Software Setup Instructions

3.2 Site Usage and Functionality Guide

References