Alabama Liquid Snake

University of Pretoria

Epi-Use

Botic - Privacy aware chatbot
Implementation Details

Justin Grenfell - u16028440
Peter Msimanga - u13042352
Alicia Mulder - u14283124
Kyle Gaunt - u15330967
Lesego Mabe - u15055214

Contents

1 Introduction

This documents follows the implementation decisions made during the project development stage. Implementation decisions will be covered in each interation, and categorized further according to the use cases.

1.1 Interation 1

1.1.1 UC21: Login

1.1.1.1 Dealing with the MVC

The Angular SPA application framework does not necessarily have a direct mapping to our architectural designs and decisions; as to be expected. The View will be bundled into varous Angular components. The Controllers will be bundled into various Angular services. The Model will also be represented as Angular services. This is how we will map each of the Angular constructs to the architecture we have chosen.

1.1.1.2 Angular and Abstract Classes

The Design Class Diagram represents certain constructs that are not necessarily Angular or Typescript contructs. Since we have made the decision to represent all our controllers as Angular services, we will have to create an Interface to represent the Abstract Controller class. Each Concrete Controller will thus be represented as an "implemented" Angular Service; [1] provides a tutorial on how to do this.

1.1.1.3 Updates to File Structure

The "shared" folder will contain classes and objects that are shared across the system, i.e. log classes, data structures, and others.

1.1.1.4 Database Manager

An Angular Service will be created to interface with the Database Manager API; this is done to decouple all networking away from each use case controller that will need to store or retrive any data or objects to and from the available databases.

The database manager itself was developed to be a RESTful api using the help of the tutorials in [2], [3], and [4].

Since we have already begun the use of Typescript in our Angular frontend, we can continue using it in our Node API. The benefits of this are getting a more "universal" language and standard for our project, including testing, having less errors because we now use a type safe language rather than JavaScript, Typescript can allow our IDEs to expose project modules easily and the more robust language offers more reliability[**?**]. To this end, the tuturial in [**?**] helped us use TypeScript instead of JavaScript in our RESTful Node API.

1.1.1.5 Session Implementation

LocalStorage has it's advantages and disadvantages, particularily that it is vulnerable to Cross Site Scripting attacks. etc etc we have chosen to use Cookies instead...motivation...motivation

But we however, use the bridge pattern to decouple how the session is stored. One can change this from Cookies to LocalStorage this way.

2 References

[1] G. Marlow, “Creating interfaces for angular services,” https://medium.com/hackernoon/creating-interfaces-for-angular-services-1bb41fbbe47c, October 2017, accessed on 2019-08-04.

[2] A. Adelakun, “Build a restful api with node.js and express.js part one,” https://medium.com/@purposenigeria/build-a-restful-api-with-node-js-and-express-js-d7e59c7a3dfb, April 2018, accessed on 2019-08-06.

[3] ——, “Build a restful api with node.js and express.js part two,” https://medium.com/@purposenigeria/build-a-restful-api-with-node-js-and-express-js-part-two-3d7a82b8e00, April 2018, accessed on 2019-08-06.

[4] D. Inyang-Etoh, “How to build simple restful api with nodejs, expressjs and mongodb,” https://medium.com/@dinyangetoh/how-to-build-simple-restful-api-with-nodejs-expressjs-and-mongodb-99348012925d, July 2018, accessed on 2019-08-06.