CS673S18 Software Engineering
Silver Serpents - Projects Portal
Software Design Document

Team Member	Role(s)	Signature	Date
Ben Schwartz	Group Leader	Ben Schwartz	03/28/18
Chad Cover	Configuration Leader Environment and Integration Leader	Chad David Cover	4/26/2018
Andy O'Connell	Implementation Leader	Andy O'Connell	
Kanishka Bagri	QA Leader		
Prabhakar Punj Lal	Design Leader		

Revision history

Version	Author	Date	Change
1	Ben	3/28/18	Entered member information and signed my name
2	Ben	4/12/18	Added introduction
3	Andy	04/18/18	Wrote about front-end classes, methods, and algorithms
4	Ben	4/25/18	Finished introduction
5	Chad	4/25/18	Wrote about back-end classes, methods, and algorithms

Introduction

This document is our Software Design Document. We will begin by describing our project and what we aim to do with it. We will also give a detailed description of the project's software architecture and breakdown the purposes of our many components, and their relationships to each other. We will add a diagram for better visualization of how the components relate and work together. This document will also contain an explanation of our most important algorithms, methods, and classes.

We have named our SPA "Project Portal" and envision it being used by Professor Zhang and her future students to access group projects. The Project Portal will house the many projects of both the current and previous semesters. People can put projects into the portal where they can be viewed and searched. People using the portal will be able to register or be guests. As a guest, users can search for uploaded projects using keywords or titles and view those projects; however, guests are not allowed to add, delete, or edit projects. In addition to the ''Continue as guest'' button, there is also a ''Create New Account'' button. After pressing this button, the user will be directed to the registration page. Here, the user must provide a name [first and last], a Boston University email address, and a password [of at least 8 characters]. If all the conditions are met, the user is registered and his/her data is stored in the database (but not before the password is hashed). Once a user is registered, s/he can go back to the home screen enter his/her email and password, then click ''Login'' Once logged in, a user can add, edit, delete, and search projects.

Software Architecture

Our project is based upon the MEAN stack, with the most important technologies being MongoDB, Express, Angular, and NodeJS. Code is written either in Javascript or Typescript and compiled into EcmaScript 6. MongoDB is our document database. Express is the Controller middleware, taking care of both the routing control logic. Angular is the View, the front end framework that the user sees. And NodeJS runs the back end of the platform, serving the application.

The software employs a MVC. The Model is implemented using Mongoose, an ORM that enables object mapping to the backend database, which is MongoDB. The backend project contains a single folder, /models, which contains all of the Mongoose models. The Mongoose models define a Mongo schema, using strong typing, and are exported as components, which are then imported by the frontend to instantiate and manipulate Mongoose objects.

The Controller is implemented in Express, which enables routes to be mapped to specific controller methods, all of which can be found in the ./controllers folder in the backend project folder.

The View is implemented in AngularJS, with the styling done with Material-UI, and code written in Typescript and compiled into EcmaScript 6. Views are composed of one to many components, with each component contained in its own folder along with the html template, style, unit tests, and component controller. Components may instantiate and change object states by calling services that map to the Express controllers in the middleware.

Each part of the system contains the three models in the project. This allows the front end and back end to easily manage the information created by a new user or retrieve something from the database.

The basic folder structure of the backend is as follows:

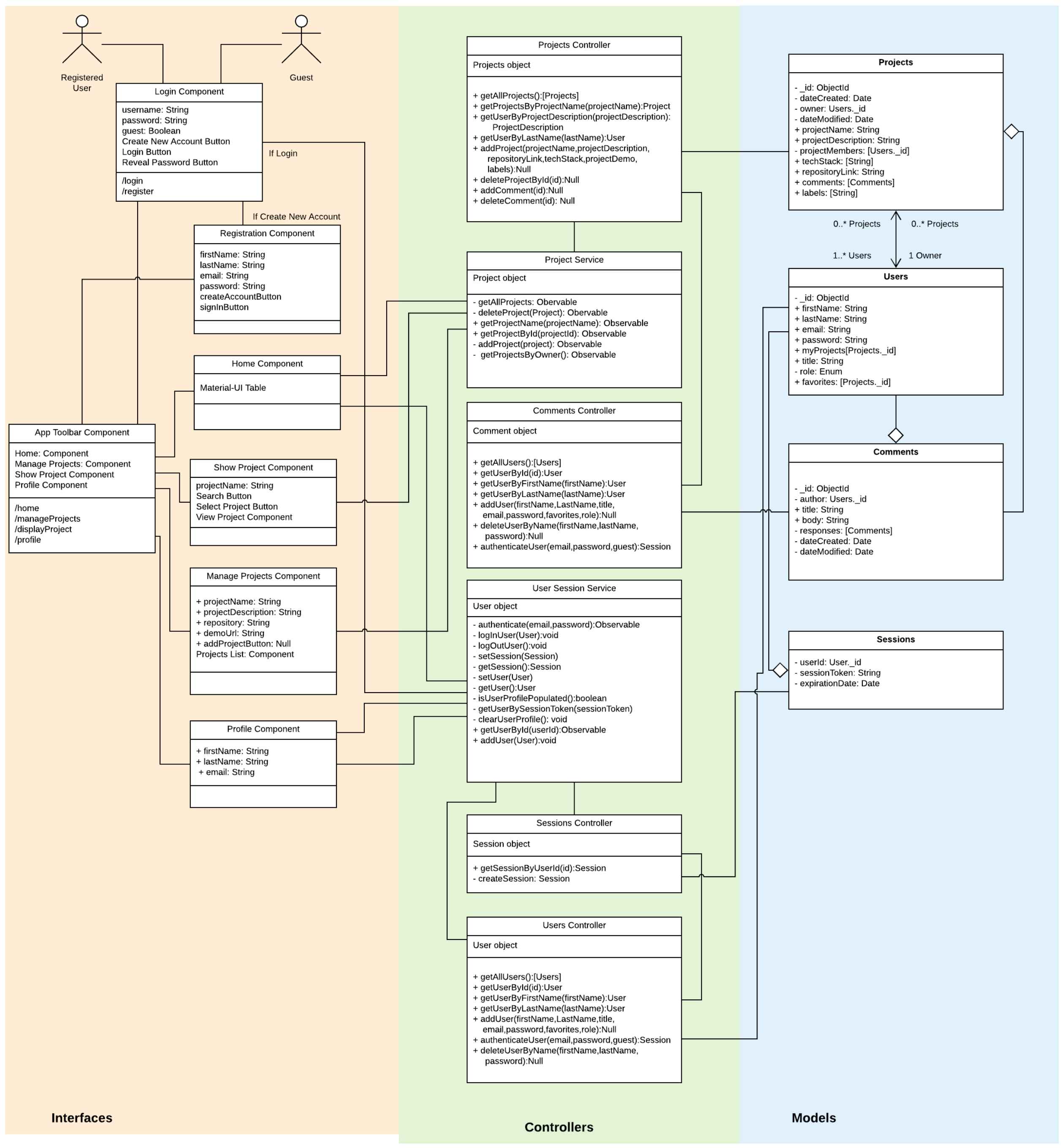
```
project
|  README.md
|  app.js
|  package.json
└──controllers
|    | users-controllers.js
|    | sessions-controllers.js
|    | projects-controllers.js
└──models
|    | users-model.js
|    | sessions-model.js
|    | projects-model
└──config
|    | default.json
|    | prod.json
|    | test.json
└──project-angular-src (the frontend folder)
```

The View is implemented in AngularJS, with the styling done with Material-UI, and code written in Typescript and compiled into EcmaScript 6. Views are composed of one to many components, with each component contained in its own folder along with the html template, style, unit tests, and component controller. Components may instantiate objects by calling front-end models that, in turn, subscribe to services that call Express controllers in the middleware.

The basic folder structure of the frontend is

```
project-angular-src
|  README.md
|  package.json
|  tsconfig.json
└──src
|  └──app
|  |  └──add-project
|  |  |  |  add-project-component.css
|  |  |  |  add-project-component.html
|  |  |  |  add-project-component.spec.ts
|  |  |  |  add-project-component.ts
|  |  └──display-project
|  |  └──home
|  |  └──login-page
|  |  └──models
|  |  |  |  Comments.ts
|  |  |  |  Project.ts
|  |  |  |  Session.ts
|  |  |  |  User.ts
|  |  └──(several other components)
|  |  └──services
|  |  |  |  project.service.ts
|  |  |  |  users-session.service.ts
```

Each part of the system contains the three models in the project. This allows the front end and back end to easily manage the information created by a new user or retrieve something from the database. The Entity, Controller and Boundary Class diagrams and relationships are as follows:



Design Patterns

Front End

In Angular there are many design patterns put to use and the framework wouldn’t work without them. First is the Observable, an entity that maintains a list of dependents and when a state changes it notifies the dependents. This is used extensively in Angular to account for inconsistent response times from mongodb. It allows REST calls to be processed on the front end when something is returned. This also means that the code is non-blocking and the website won't freeze when one rest endpoint is stuck.

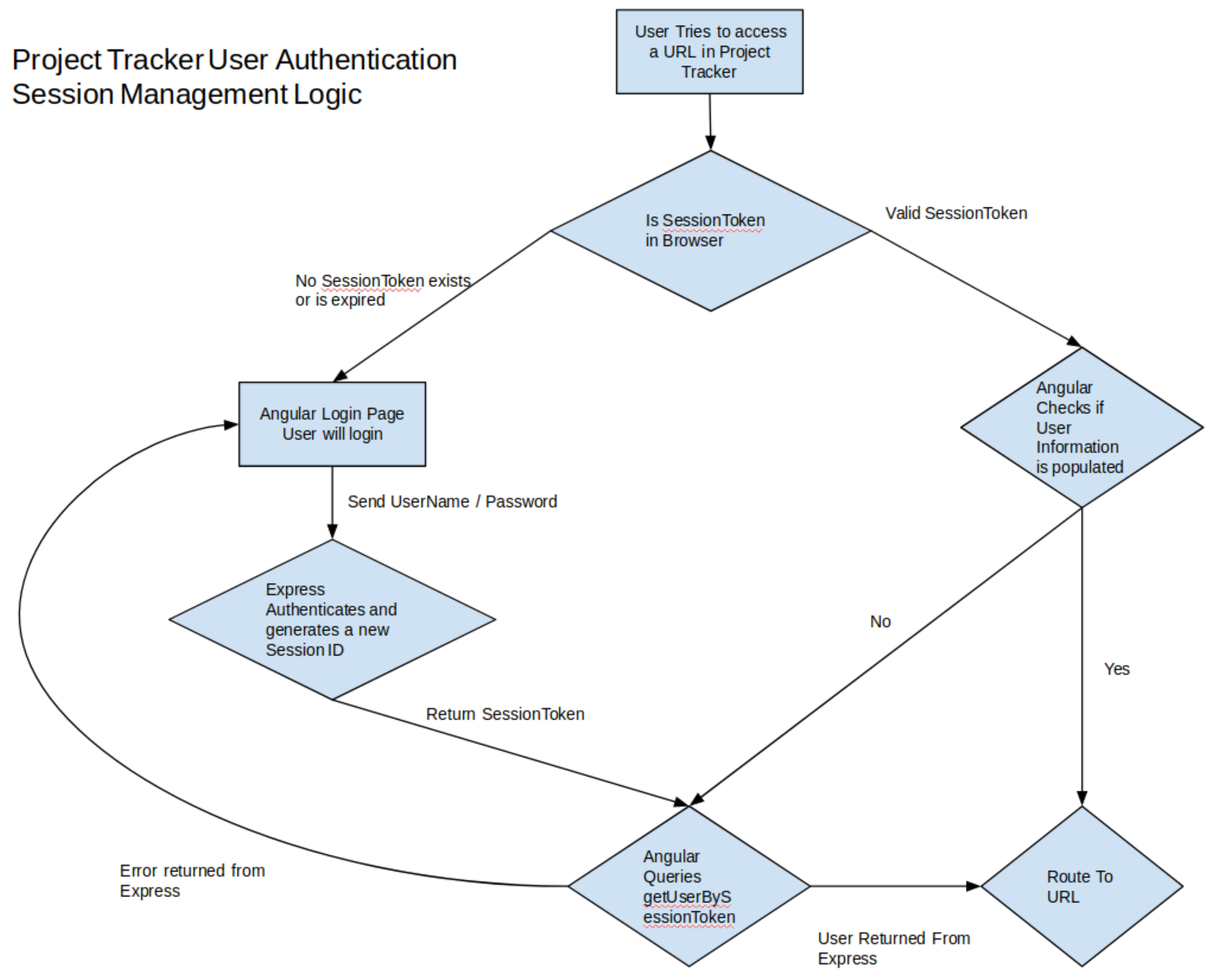
The Composite design pattern is used for the Components in Angular. All components are treated as the same object and depending on the routing of the webpage all the components are dynamically generated together. This allows a compartmentalization of business logic on the front end. For example If the designer wants to reuse the navigation bar found at the top of the screen, she only needs to make one navbar component that can be reused across the rest of the project.

Back End

Mongoose employs the Facade pattern, offering simpler interfaces to the more complicated set of APIs used in MongoDB.

Key Algorithms

For Authentication we use a mix of SHA256 and state storage in the browser. We use authentication because we do not want to allow anonymous users to create or modify projects.


Project Tracker User Authentication
Session Management Logic
User Tries to access a URL in Project Tracker
Is SessionToken in Browser
Valid SessionToken
No SessionToken exists or is expired
Angular Login Page User will login
Angular Checks if User Information is populated
Send UserName / Password
Express Authenticates and generates a new Session ID
No
Yes
Return SessionToken
Error returned from Express
Angular Queries getUserBySessionToken
Route To URL
User Returned From Express

At the entry point we check to see if there is a session token for that user.

Classes and Methods

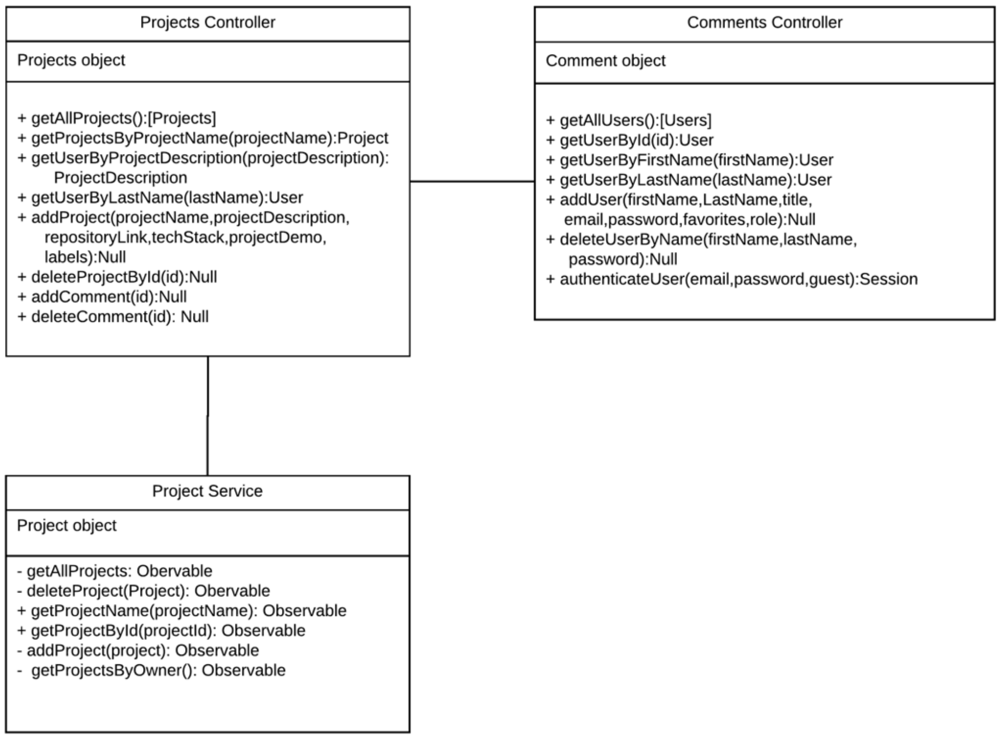

Projects Controller
Projects object
+ getAllProjects():[Projects]
+ getProjectsByProjectName(projectName):Project
+ getUserByProjectDescription(projectDescription): ProjectDescription
+ getUserByLastName(lastName):User
+ addProject(projectName,projectDescription, repositoryLink,techStack,projectDemo, labels):Null
+ deleteProjectById(id):Null
+ addComment(id):Null
+ deleteComment(id): Null
Comments Controller
Comment object
+ getAllUsers():[Users]
+ getUserById(id):User
+ getUserByFirstName(firstName):User
+ getUserByLastName(lastName):User
+ addUser(firstName,LastName,title, email,password,favorites,role):Null
+ deleteUserByName(firstName,lastName, password):Null
+ authenticateUser(email,password,guest):Session
Project Service
Project object
- getAllProjects: Obervable
- deleteProject(Project): Obervable
+ getProjectName(projectName): Observable
+ getProjectById(projectId): Observable
- addProject(project): Observable
- getProjectsByOwner(): Observable

Figure 1. Projects and Comments Class Models

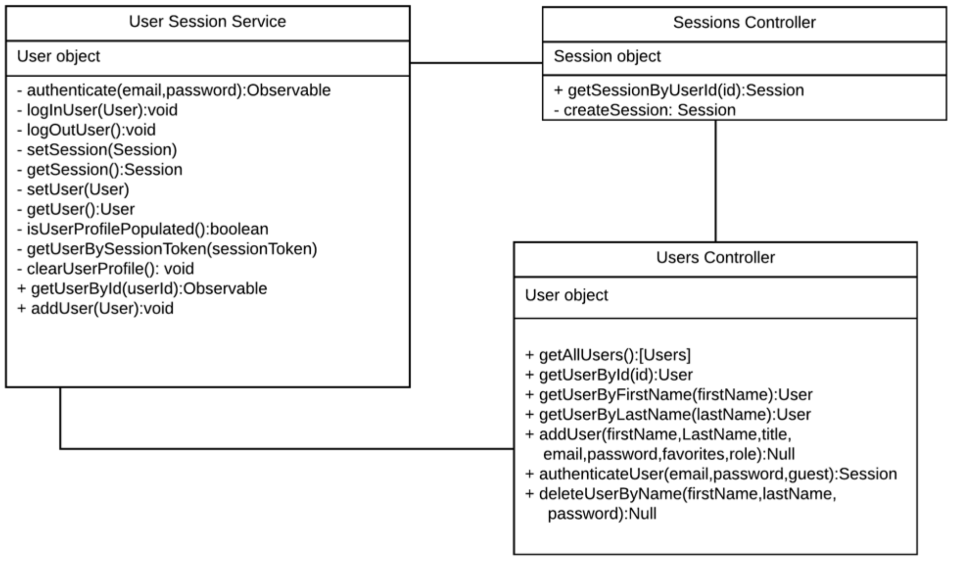

User Session Service
User object
- authenticate(email,password):Observable
- logInUser(User):void
- logOutUser():void
- setSession(Session)
- getSession():Session
- setUser(User)
- getUser():User
- isUserProfilePopulated():boolean
- getUserBySessionToken(sessionToken)
- clearUserProfile(): void
+ getUserById(userId):Observable
+ addUser(User):void
Sessions Controller
Session object
+ getSessionByUserId(id):Session
- createSession: Session
Users Controller
User object
+ getAllUsers():[Users]
+ getUserById(id):User
+ getUserByFirstName(firstName):User
+ getUserByLastName(lastName):User
+ addUser(firstName,LastName,title, email,password,favorites,role):Null
+ authenticateUser(email,password,guest):Session
+ deleteUserByName(firstName,lastName, password):Null

Figure 2. User and Session Class Models

The front end is using several models to allow us to connect to the backend Express server.

Angular contains the following models:

- User
- Session
- Project
- Comments

Each of these Models is related to their Backend peer. In the Angular project all of the modes are contained in one file and there is a Typescript file for each model. The directory is /src/app/models/. These models are then imported into the necessary components throughout the angular app. An example of this is let's look at the login page. This would be the first page the user would normally see.

References

- Mongoose - http://mongoosejs.com/
- Express - https://expressjs.com/
- AngularJS - https://angularjs.org/
- NodeJS - https://nodejs.org/en/
- Material-UI - https://www.material-ui.com/#/
- Typescript - http://www.typescriptlang.org/

Glossary

- MEAN - an acronym for the stack [group of languages] is MongoDB, Express.js, AngularJS(or Angular), and Node.js [Wikipedia]
- ORM - (Object-Relational Mapping) - is a programming technique for converting data between incompatible type systems using object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language [Wikipedia]
- Mongoose is an object data modeling (ODM) library that provides a rigorous modeling environment for your data, enforcing structure as needed while still maintaining the flexibility that makes MongoDB powerful. This article shows the use of Mongoose in a simple Node.js application on Heroku. [Google]
- ECMAScript is a trademarked scripting-language specification standardized by Ecma International in ECMA-262 and ISO/IEC 16262. It was created to standardize JavaScript, so as to foster multiple independent implementations.[Wikipedia]
- REST - REpresentational State Transfer (REST) is an architectural style that defines a set of constraints and properties based on HTTP. Web Services that conform to the REST architectural style, or RESTful web services, provide interoperability between computer systems on the Internet. REST-compliant web services allow the requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of stateless operations. Other kinds of web services, such as SOAP web services, expose their own arbitrary sets of operation [Wikipedia]
- MVC - The Model-View-Controller (MVC) architectural pattern separates an application

into three main components: the model, the view, and the controller. The ASP.NET MVC framework provides an alternative to the ASP.NET Web Forms pattern for creating Web applications [Google].