**EventSM**

This assignment is a pair work with Abhishekh Baskaran (student number).

The webpage is available on: <https://localhost:1337>

Our website is designed for socialising, where a user can create an event or join an already existing one and chat about it in the chatroom provided. The project is based on the Sails.js framework, which is build on top of Express.js to provide MVC functionality and embedded database. For the frontend, Bootstrap framework is used, along with Backbone.js to deal with the (synchronising messages?). (Describe backbone in few words).

As functionality, such as routing, security and web sockets is provided by Sails, we have instead focused in diving deeper and developing a real-time chat interface.

The directory structure is as follows. The *api* folder contains controllers, models, helper scripts and policies. *Assets* folder is home of the custom client-side js files, images and css styles. All configuration files that ensure the backend is working correctly are contained in the *config* directory. Most of them are Sails.js default and were not modified, or only altered to extend the functionality beyond the basics provided. *Tasks* folder contains scripts that are used whilst developing the application and was not modified. These are run by Grunt, the javascript task runner, that comes with the framework by default. Web pages and the layout, which all pages inherit are stored in the *views* folder. The default error 404 and 500 pages were not modified.

To ensure that javascript syntax is correct and it follows convention the ESLint plugin for Atom editor was used. We used the Ejs templating engine for creation of dynamic pages, that are translated into HTML files as the server is started. A layout was used, to ensure all pages have the same basic structure and XHTML was enforced in the layout.ejs, which all pages inherit.

A ssl certificate was created and the application is now accessible via the https secured connection. The certificate is stored in the config directory and configured in the local.js. All routing that handles GET and POST requests is defined in the routes.js.

The *Bootstrap* framework was used to design the frontend along some custom styles, for the navigation bar, footer and the error div that gets displayed whenever the login/signup is unsuccessful. In addition, there is a simple fade out animation of the error div defined in the clientjs.js file. To display two Github icons in the footer of the website, that lead to our Github profile pages, the *FontAwesome* library was used.

Responsive design is ensured with Bootstrap, (i.e. the navigation bar changes into a dropdown menu), although we targeted desktop and only checked the mobile device compatibility via the Chrome debugging tool. As we are using the latest version of Bootstrap our website supports all modern browsers, but it no longer supports the IE9 and its earlier releases.

The background photo was obtained from Unsplash (<https://unsplash.com/>) but was modified extensively using Adobe Photoshop. The saturation and shadow brightness were increased as well as minor contrast and sharpening changes were done on the photo. At the final stage the horizontal axis was adjusted, and the photo was resized to 1920x1080 resolution and converted into a PNG format.

SVG

**User Guide**

After visiting the homepage, user first needs to login or create an account via the ‘Log in’ and ‘Sign up’ options. The authentication is done using the *Passport.js* library but written from ground up with the use of their documentation. Signup and login calls are handled with the AuthenticationController, whilst the core authentication is done in the passport.js file.

As Sails.js comes with ORM called *Waterline*, there is no need for writing SQL queries to set up and interact with the embedded database as the framework offers the functionality by default. Some basic form validation, such as checking if an email has a valid form or if the username is unique, used in the signup stage, is provided by the Sails.js framework. Validation requirements are set along the definition of the user model in the User.js file. The model is automatically added to the database at the start of the server. The password is also hashed before storing, this is done with *bcrypt* library, and custom failure messages are displayed to the user if the signup/login is unsuccessful. A simple isLoggedIn.js policy is defined and configured in the policies.js, which restricts calls to only users that are authorised.

Once the user is authenticated it is redirected to the homepage, where it can now access Events and the Chatroom tabs.