Design:

The project is split into two parts, the frontend at the /front/ folder and backend at the /back/ folder. The frontend uses AngularJS and split into multiple folders each detailing an "area" with associated controllers for itself as well as any modals it may need. A singular nav-bar is injected into every page except for the landing. The admin panel can be used to view all information as well as clearing the database. The different views are usually kept separate communicating only through the models in the database. However in some cases when data needs to be passed back and forth, the dataBus service is used. The dataBus service is used by setting a value then getting the value later, however the value stored can only be gotten once as when "get" is called, the value stored is cleared before returning. An additional auth service is used to keep track of the current user. On the backend, the REST endpoints mostly reside in the api file and all endpoints are prefaced by "/api". The user-controller defines interactions with the user, defining gets as well as get many's, the same applies to the event-controller. Get operations involve looking at the query for search filters and sort definitions. Additional endpoints are found in the '/dev' prefaces in the dev route file. These endpoints are for development and admin use to clear users and events as well as populate the database with fake data.

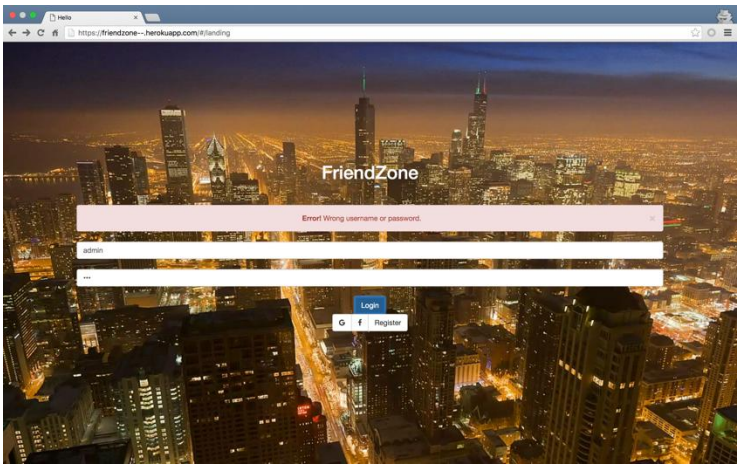
Security:

We implemented security for our sites through a number of ways.

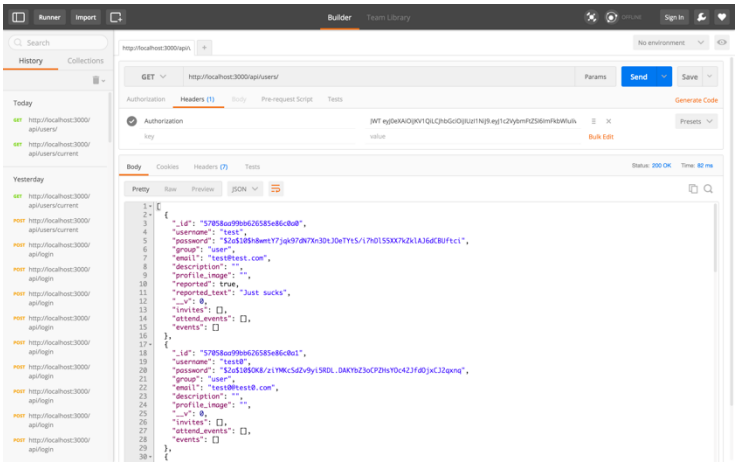
First, for every user, we use a module called bycrpt to hash the password before storing it in the mongodb database, making sure it is not stored in plain text. Before each save() call, we check if password has changed; if it is, we hash it and overwrite the old password and save in the database.

Second, we use an authentication middleware called Passport for Node.js + Express, in order to protect our RESTful api endpoints. For logging in and signing up, we utilize Passport Local strategy that authenticates a username and passport combination. After a successful authentication, we create a JSON Web Token and assign it to the user; then we use Passport-JWT strategy to authenticate subsequent API calls using this token. At the front end, we have a AngularJS injector module that sets the Authorization header with the JWT token generated earlier. We also provide a HTTP Basic Authentication strategy for our API endpoints, and our back-end accepts both methods.

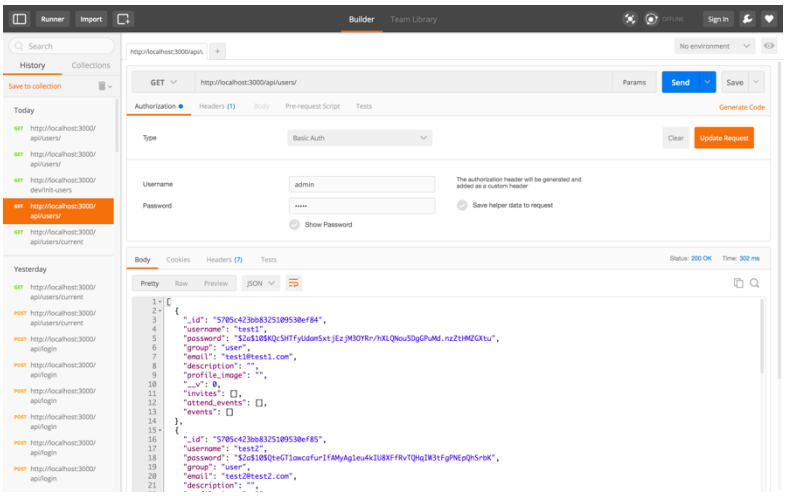
Below are some screenshots:



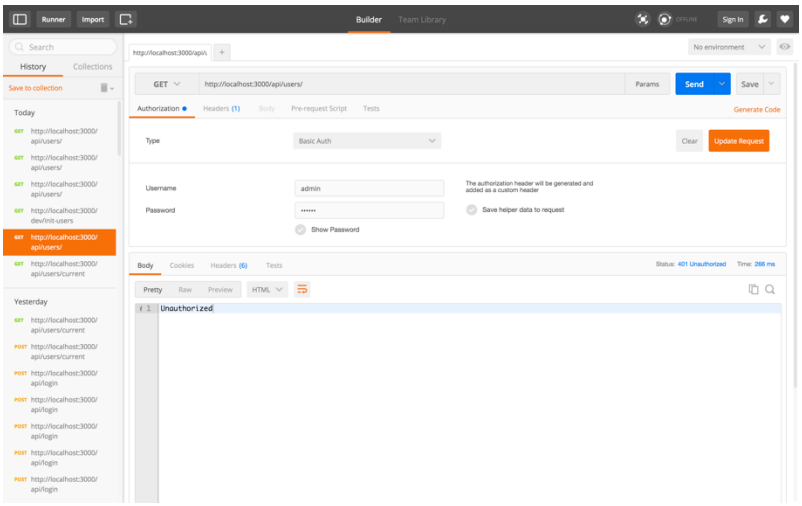
Notice that when a username/password combination is incorrect, an error message will appear and the user is prompted to retry.



Using Postman to test API endpoints. Authorization header is set to the JWT token and we are authenticated to perform a GET request to retrieve all users.



Using Basic Authentication by providing a username and password also works.



If authentication failed, server responds with a 401 (Unauthorized).

Mocha was used to test the functionality of all pages and functions for correctness. These tests attempted to retrieve useful information from the page functions including verifying the creation of users and events. Other tests also tested for the returns of the search function and navigation functionalities.

Optimization:

We used locust.io to simulate 300 users at 10 second intervals to test loading times of our landing and profile pages. This generated no failures or exceptions.

The resulting statistics:

Method	Name	# requests	# failures	Median response	Average response	Min response	Max response	Average Content	Requests/s

				time	time	time	time	Size	
GET	//landing	3693	0	7	8	3	96	27	29
GET	//profile	1817	0	6	8	3	75	27	14.27
None	Total	5510	0	7	8	3	96	27	43.27

The time distribution chart:

Name	# requests	50%	66%	75%	80%	90%	95%	98%	99%	100%
GET //landing	3340	6	8	9	9	13	17	23	31	96
GET //profile	1639	6	8	9	10	13	17	24	30	75
None Total	4979	6	8	9	9	13	17	23	31	96

Video:

Located at this link: https://youtu.be/IvTcBKPKGyY