# Backend Work

## Express and MongoDB

**Recap from Project 2**

From project 2, there were only several APIs implemented that served the basic functionality of the web app.

1. Admin Sign ups
2. Admin Log ins
3. User requests from outside the app
4. Admin viewing all the messages
5. Admin seeing more details of a particular message

This was the prototype of the app that was shown to Kenneth (Our contact from Accenture). He then gave the feedback that we should have more “ticketing features”. He also posted a general message on the Slack chat that it would be good for users to log in to view their requests. Hence, this gave us several ideas of things to implement for project 3.

**Project 3 updates**

*  
Table 1: User management APIs (this was associated with 2 collections on MongoDB)* *Table 2: Request Management APIs*

Following up on our discussion with Kenneth, our team decided to split the database into 2 collections, the USER\_MANAGEMENT collection and the REQUESTS collections. On the backend side this would make handling of data easier as modularity is increased. We then added a host of APIs to add additional functionality. A detailed table of APIs can be found in Table 1. In summary, some of the more notable features include

1. Adding a generic sign up page that can differentiate users from admins based on their email addresses
2. Encrypting user’s and admin’s passwords so that security is improved
3. Adding a key word extractor to extract tag from user posted messages
4. Adding a sentiment analyser that detect sentiment from messages and provides more effective ticketing
5. Adding various APIs for filtering the messages based on date, status, category, sentiment and who was addressing it
6. Adding a chat API that enabled users to chat with admins

These features were demonstrated to Prof Sudipta during Project Meeting 3 on 22 March 2019. The notable features was the encryption feature which fulfilled a non-functional requirement by Accenture (They wanted password security). Also, having ML features like key word extraction and sentiment analysis were also non-functional requirements and bonus features that our clients would like.

In addition, we also demonstrated to Prof that the server was successfully deployed on Heroku which ensured that that the app was running online. This meant that team members could do integration without having to download the backend code and running it on localhost.

Furthermore, we also decided to use Mongoose to interact with the database instead of using MongoDB functions directly. This had the advantage of allowing us to define the schema without logging into MongoDB and pre-defining it. Thus, it was easier to come up with the schema and configure the data structures associated with it.

## Backend Testing

  
Table 3: Test cases for the user management APIs

  
*Table 4: Test cases for the Request APIs. Some of these tests would be better to test with front end testing as it uses object id which is handled and kept constant by the server*

As per what we were taught in class, it would be prudent to test every use case that was implemented. We focused our testing using a mix of black box and white box testing. The black box tests were more direct in nature as we tested more normal instances of the app (Normal signup, login, posting messages, viewing requests). On the other hand, for the white box testing, since we knew the code, we could come up with boundary test cases (Username empty, wrong id sent during the viewing of messages). Figure 12 shows one such test for user (without accounts) posting messages on the app. The testing for backend was done with Mocha and Chai and tested the API end-points that were deployed on Heroku.

*  
Figure 12: Some test cases for user signups specifically the cases where users have submitted an existing username and the case where they have left their emails blank*