Team: MSDAPI

Team Members: Akshay Kulkarni
Aravindhan Eswaran
Yang Song

Project Team Report

Frameworks Used : Node.js
Database : MongoDB
Testing Client : Mocha
Library for Client : REST library

Before deciding on the framework for each the above mentioned sections, we researched on various aspects and discussed the pros and cons of each of them. We then arrived at the design philosophy to use Node.js coupled with MongoDB and test using Mocha and Write clients using REST library.

Our first task in the project was seamless deployment. So, the initial setup was integrating Bitbucket with Codeship and integrating Codeship with Heroku. This proved as a challenge to the team as none of the team members had used any of the services before. Once we got all the configuration right, we got our first build (green build). After the '**Hello World!!**' build on heroku, we then included MLab to our heroku deployment.Our initial weeks went ahead discussing the design choices which we had to make. On the high level design we broke the problem statement into smaller modules dealing with

- User
- Pet
- Comments and Questions.

This way visualisation of the tasks became more easier. For example, for anything related with Pet, we decided to have an admin check, so as per our design we just needed to include this check in the services which included pets. These design decisions proved helpful to us as we were able to break down complex service tasks into simpler design patterns. Then, we created a shared document in which, we would write the name of the module and the APIs in the module to be created. This documentation also included the data format which will be used and the contents of the key value pair. After that we would split these tasks and assign the corresponding tasks to each person in JIIRA. It was then the responsibility of that member to finish his part. Once done, before the stand up, the team members would meet and discuss our progress and difficulties. Then based on team member's suggestions, we would modify or change the code to make it more efficient. The TA was satisfied with how we carried ourselves as a team through the development process and mentioned that we were moving good as a team and were also progressing faster compared to other teams. This served as a motivation for our team.

Team dynamics

- Did you work well together?
    - Yes, we worked very well as a team. Our wavelengths matched on all levels. We never had issues working with each other. We gelled very well as a team and functioned as one. Even when we had issues to deal with we never pointed fingers at each other and blamed

for any shortcomings, instead we discussed how an issue can be resolved and made sure the issue at hand is resolved before we move forward.

- Did you have challenges to overcome?
    - Implementing the code was not such a big challenge as integrating Swagger for documentation was. As a team we had to sit together and all the team members worked on separate methods to get the documentation working.
    - Integrating MLab with Heroku was one of the challenge we faced. It took some time to understand and integrate it into our service.
    - We worked on three separate branches and there were bound to be merge conflicts, but we could resolve those.
    - Also writing clients for other team is always a challenge as one needs to understand how they have designed their service, what were their philosophies and why did they write their service in a particular manner.
- (Since everyone should have had some challenges) How did you overcome them?
    - For getting the Swagger documentation to work, we had a meet in which we all three worked on it and we all three tried different approaches to make it work. Our hard work paid off and we were able to integrate one of the approaches in our code and make the documentation work.
    - Before MLab(for Database storage) we used to use Mock JSON objects to test if our code worked as we expected. Also we used local databases' to test our code functionality.
    - For merge conflicts, we had dealt with similar issues before so it was relatively easier. But surprisingly there weren't much code conflicts as we had imagined beforehand.
    - For writing clients, it was again a team effort. We used the Node.js - rest library to test the services. The main challenge was to nest the clients inside each other as the other team was using an Auth which was required in every request. This made writing the clients a bit of a challenge. Also, since we were making asynchronous method calls to the PostgreSQL, we were not sure of the exact time in which we might receive the response which needed to be included in the next message. For eg.,. We needed to receive the petId from the database, before updating the pet.

Experience writing a client for someone else's service

Things that are going on well: We are fortunate to find a very convenient library for writing clients in Node.js - the rest library. It makes it a lot easier to sending HTTP requests and handling HTTP responses. Also, fortunately the web service of PizzaOrange used JSON for requests and responses, since we are using Node.js, it's also very easy to compose requests and decompose responses in JSON. In addition, it's nice that PizzaOrange provided some examples on how to properly using their APIs.

For things that are not going on very well: PizzaOrange is not using Swagger as their documentation, so we are not able to take advantage of the built-in test client with Swagger. Also when we passed in an error request to test the robustness of PizzaOrange's APIs, their service threw too many error messages. It looks like they are using a debugging or verbose mode of Django which returns all the

stack traces, that made it difficult to analyze the response. We often need to scroll several screens to find the really useful error message.

It will be very helpful if the other team could provide an interactive documentation interface (instead of just pdf manual) that we can directly use, e.g the try-it-out functionality of Swagger. It makes testing a lot more convenient.

From the experience on writing client for others, we think documentation is important, but it will be even better if we included a live demo on how to use our service, from creating an admin user, logging in and out, to posting and deleting comments. That would reduce a lot of confusions for other users.