Blore, Nathan

nathan.blore@ada.ac.uk

Abstract

Evaluation on my End Point Assessment   
for the membership system I created for  
Bows Formula One High Performance   
Cars.

EPA Evaluation

Table of Contents

[Introduction and Scope 2](#_Toc31380943)

[Tools & Languages Used 2](#_Toc31380944)

[Criteria 3](#_Toc31380945)

[Limitations and improvements 4](#_Toc31380946)

[User Guide 5](#_Toc31380947)

[Conclusion 5](#_Toc31380948)

Introduction and Scope**:**

My customer, First Catering Ltd, has received a contract from Bows Formula One High Performance Cars, to enable users of their business to use their existing employee cards at a kiosk to view their balance and top up their balance.

I had to design and build a RESTful web service API for the kiosk terminals that will be used to complete the requirements above.

Tools & Languages Used**:**

**Java:** I decided to write in Java due to the fact I have the most experience in developing with this language. I have previously created a similar system for a different project which was used to onboard and offboard users. I decided that instead of using a language I was less familiar with I could take my experience and knowledge from the time I created this application for my Work Based Project.

**Springboot:** This is a Java framework used to help create applications. It makes it easier to create and run Java applications. Similar to above, I used Springboot to create the onboarding system I used for my Work Based Project previously so decided that it would be the best framework to help me create my application that connects with my database.

**IntelliJ:** I used this IDE to write all my code in because I have the most experience using this. I have developed in IntelliJ for several years now and know a lot of the shortcuts and useful development features it comes with. It also has lots of plugins and features that help aid my development, such as specific code formatting and file preview options.

**MySQL:** MySQL is a relational database system and the thought process behind deciding to use MySQL on this project was the fact that I again ended up using this for my Work Based Project, so I did have some prior experience with this.

**MySQL Workbench:** This was very helpful as it allowed me to view the database in a visual format on a GUI, graphical user interface. This wasn’t required however did make my life a lot easier when I wanted to check the database for data as I didn’t have to view it through the command line instead. I also did use this on my Work Based Project as well as using it at my workplace to view databases and run queries, so I knew how to use it beforehand.

**Git:** I used Git to help me with my version control. For example, there were moments I wanted to make changes to my code without breaking everything else I had already done. So, I would create a branch off my current one just incase I did need to revert back to what I had previously made.

**Github:** Linked with Git, I used GitHub as my place to store all my files for my End Point Assessment. Everything is available through the github link (<https://github.com/nblore/epa>). It is an easy way for me to keep all my files together and means that they are easily downloaded. It also displays the README (my user guide) to anyone who accesses this link.

**Postman:** Postman was the recommended piece of software to use for API testing. Since my entire project was based around the API endpoints, I decided to go with the recommended application to help me with my project. It was very easy to use and user friendly and helped me test my endpoints acted as expected

**iTerm2:** iTerm2 was my version of terminal that I liked to use as I needed to use this to start my springboot server and to push my code on GitHub. I have used iTerm2 for several years now and it is a much more user friendly version of the built in Terminal that comes with the Mac.

Criteria**:**

As mentioned in my test plan, I did end up covering all criteria under the ‘Business Requirements’ page in the brief. Below I have listed how each endpoint for my API are intended to be used by the kiosk clients and to prove each requirement is met.

*http://localhost:8000/api/employee/login*

This endpoint is used for a user to log in once they tap their card. Once they tap it twice it will also be used as the log off endpoint.

*http://localhost:8000/api/employee/register*

This endpoint is used when a user has not registered their card before and they will then be instructed to fill out each field (employee ID, name, email and mobile)

*http://localhost:8000/api/card/balance*

This endpoint is used when a user selects “View Balance” after logging in. This endpoint will return the users balance, as they will already be logged in. This endpoint will not return a balance to a user who is not logged in or whos session has expired.

*http://localhost:8000/api/card/topup*

This endpoint is used when a user selects “Top Up” after logging in. This endpoint will top up the users balance depending on how much the user inputs when wanting to top up their account. This endpoint will not return a balance to a user, or allow a user who is not logged in, to top up their account.

*http://localhost:8000/api/employees*

This endpoint will return a list of existing employees in the database when a manager clicks “View All Employees”. This endpoint will only be available to be reached from managers and not any employee.

*http://localhost:8000/api/employee/delete*

This endpoint will delete a user based on the employee ID that the manager inputs after clicking “Delete User”. It will update the database and the employee will no longer exist in the records. This endpoint will only be available to be reached from managers and not any employee.

Limitations and improvements**:**

The biggest limitation was time for me. I only had 5 days to complete this task and I had not used Java since my work based project so I had to spend some time familiarising myself with it beforehand. Because of this I am not convinced the code is of the best quality it could be and that it might look a bit messy and ugly in some parts.

I also struggled with the fact that there was no frontend interface required on this project. I did not like having to use the backend to complete every requirement listed. For example, I would usually implement the timeout feature on the frontend system as it would be a lot cleaner and would make more sense however I had to instead take a different approach and do this in Java and MySQL. I had to create an extra field for the employee table which seemed like a lot of extra work that would not have been needed if I instead had a frontend interface that I could work on.

I also wanted to include multiple HTTP requests, as mentioned in my design brief. I created the project following all the business requirements and realised I had only used POST requests. So, I added the option for a manager to be a different user type so they could use a GET request to list all employees as well as a DELETE request to delete an employee who no longer works there.

The application is fast and returns information quickly, however I am sure that I could speed this up further but due to time constraints my main objective was to get the system working and if I had time I would look at speeding the API call up afterwards.

My code follows good coding standards and I have used the framework correctly. I have multiple classes, a folder for all of my code as well as having a separate folder for where my test cases would be written had I had more time. I also have my packages together so the project is very easy to navigate around.

I would improve this project, or make changes to the requirements, if I had more time/was allowed to by adding a frontend. A frontend would make it a lot easier for me to test my code as well as allowing me to include some of the requirements in the frontend instead of having to do it all in Java and MySQL.

If I had more time I would also write Junit tests to help automate my testing process and find issues with my code earlier. I know my code does everything it needs to do however unit testing would allow me to ensure there were no errors with the code that I wasn’t able to pick up by manually doing regression testing instead.

User Guide**:**

Please see the README.md file on GitHub to view the user guide. It explains how to create the database, setup the server as well as reaching all the endpoints of the API.

Conclusion**:**

In conclusion, I did enjoy making this project and I am comfortable with my ability to create the system according to it’s requirements. However, I would have liked to do some of it in a different way but due to lack of time and the requirements asking for just a RESTful API, and no frontend, I had to compromise. Using the backend for everything isn’t best practice but was required for this project.