Project Overview

The Software Engineering department at Rochester Institute of Technology is growing at an increasing rate each year. While this is good news for the school, it does present challenges for the facilities here that support the department. One such facility is the team rooms used to organize student teams for work on projects. The expansion of the program means that rooms fill up quickly and are not always accessible to students on demand.

The department needs a way for students and faculty to be able to create reservations for rooms in advance as well as a way to configure such a system. This project is meant to help mitigate the influx of new students each year and their need of the team rooms. This would show the available rooms to those using the app and allow the students to know when a room is in use or available at a particular time.

The full scope of this project is to provide a useful tool to help both students and professors keep the team rooms organized through continually giving the user information on the rooms and their availability. This would function alongside two other portions of the app where a lab monitor or labbie, can keep an eye on the rooms’ status from a single monitor as well as see alerts given by students to tell the labbie about possibly being late. The final portion is an administrative portion to allow for the configuration of user settings and privileges.
Basic Requirements

- Users shall be able to log in with their Software Engineering credentials.
- Users shall be able to cancel their own reservations.
- Users shall be able to view all of their currently active reservations.
- Faculty shall be able to create Course Teams.
- Faculty shall be able to create a reservation that can contain multiple rooms and/or recur on one or more days for a number of weeks.
- The faculty shall be able edit what days of the week, which room numbers, number of weeks, and the end date they desire.
- Labbies shall have a view that shows that status of all current reservations.
- Labbies shall be able to retrieve notifications from users that are viewable in their view.
- Admins shall be able to create permission classes.
- Admins shall be able to configure system properties.
- Members of a team shall be able to create a reservation for a team room.
- Senior project teams shall be allowed to create reservations up to a month in advance by default.
- The system shall allow the reservations if the preemption number of the user is higher than that of the conflicting reservation.

Constraints

This project didn’t have any development constraints provided by the sponsor and it was up to the team to determine what was necessary to complete the task. Our resources were limited to the team and what could be done with thee team’s experience. The technology choice was simply an agreement among the team and with a provided VM that ran OpenBSD. This operating system did guide a few choices early on that were changed at a later point to further development.
Development Process

The Development process chosen was Feature Driven Development (FDD). This development process was determined based on the priority the features were. The system of the project was a small scale that Feature Driven Development would fit perfectly. Our sponsor approved of this development process however it was not forced upon the team. Feature Driven Development did not cover or address communication during this project. The sponsor was available through email and was able to communicate regularly in person. In this team, roles were implied and not forced. Dennis was the front end engineer, Nicholas is the back end engineer, and Dan worked on documents.

Project Schedule: Planned and Actual

The team developed a project schedule but could not follow through. The project schedule was created at the end of the first semester. The key activities and milestones were identified. These milestones were creating multiroom, adminstrator tools, view reservations, create single room reservations, labbie views, hooking up the front end to the database, requirement document, technical paper, final presentation, and reflection. As a team we tried the best to follow the schedule but realized that we were not able to deliver the milestones that we created. The milestones that were actually completed was the single room reservation functionality, Database connectivity and functionality, the requirements document, the technical paper, the final presentation, and the reflection.

System Design

The current design of this application at this point is that of a MEAN stack application, this means that MongoDB was the database used, Expressjs was implemented to manage the backend of the application, Angularjs was used for frontend functionality and templating, with Nodejs being used as a foundation for all of the JavaScript developed. Originally, MySQL was going to be used but this proved somewhat problematic for the team to implement as it didn’t function as smoothly with the rest of the stack as Mongo did. The original choice for MySQL was supplemented by the fact that the team had a bit more knowledge on how to use MySQL over any other form of database.
While the switch to MongoDB was a large change and a large risk, it improved our overall performance as it was much easier to set up and start developing the actual app instead of worrying about the database’s current schema. Unfortunately, MySQL was in fact already set up and ready to go by the time the team decided on using MongoDB instead, this led to a lot of wasted development hours as the implementation of the database was no longer used.

Fortunately, the overall design of the MySQL schema was reused to assist with setting up the newly implemented MongoDB. This led to some refactoring on the backend as the use of SQL statements and queries had to be altered to match MongoDB’s simpler queries as well as removing any extraneous queries that no longer made sense in the new dynamic schema offered by MongoDB.

The final reason for MongoDB is in fact its ability to have a dynamic schema, as this allows for the use of the same collection, which is similar to a MySQL table, to have varying key value pairs depending on what was needed when the data was added to the collection. This would allow the team to use the same collection to store both student reservations and professor multiroom reservations with ease and without having to either make a new table or design a special schema for the reservations.

The team’s choice to use a MEAN stack came from the desire to use JavaScript throughout the entire application, this was to mitigate the need to learn new languages to make the application functional between the frontend and backend. This also allowed for members working on different aspects of the project to view and understand another member’s work quickly, and determine what could be done to integrate their work into the other member’s work.

An alternative that was considered was the use of .NET, but as the group had next to no experience with C# or .NET, it was determined that using mostly JavaScript and MEAN was the optimal choice. This along with the necessary prerequisites to fully set up a .NET application would have slowed the team in the long wrong.

Since the main focus is for student teams to be using this application, single room reservations were the top priority feature. This was a key aspect of designing the architecture of the application, as it was meant to be used on mobile devices as well, so usability was a must, along with ensuring that the average user didn’t need to enter any of their own information by hand except to log in. This was in turn implemented by reducing the number of clicks it took to reach the end point of creating a reservation in addition to reducing memory load on the user. The ability to view a user’s reservations as simple as one click from the home screen was also a must as it reduced a user’s time navigating the app and allowed the user to get the information they needed.
Our focus on simplicity continued into the multiroom design as the professor was supposed to be able to easily set up and create a multiroom reservation with little need to remember each individual part of the reservation. While this would naturally require more steps to accomplish, the architecture is similar.

Design rationale for this is that it complemented itself, as Angular allows for easy and fully functional templating of html that made consistency easy to implement between multiple pages as well as supply similar development to be performed in between each portion of the application. While the multiroom portion isn’t completed, the work from the single room reservation can be extrapolated to fully finish and integrate this functionality into the rest of the application.

**Process and Product Metrics**

Metrics that the team used was the process and completed implementation of planned features. There were other metrics that the team passively used such as lines of code, number of bugs, and function points. The results of function points and number of bugs were low compared to other project systems. However, the amount of lines of code were high for a small scale project. We interpreted these results as the team sluggishly got features implemented and completed. The metrics tells the team and sponsor that the system and process went poorly to get the system completed. There were spikes of lines of code that were committed to GitHub and that indicated to the sponsor that the team had a massive spike of productivity.

**Product State at Time of Delivery**

The state of the project is that the project is overall unfinished but it has basic functionality when it comes to creating a reservation, deleting a reservation, reading a reservation, and updating a reservation. The project is missing admin functionality, which includes configuring the system as a whole; labbie functionality, which would allow the labbie to monitor an up to date view of the status of each room; and multiroom functionality, which would allow professors to create special, recurring and/or concurrent reservations for themselves or their classes. There was no unplanned features that were added to the system. However, there were unplanned requirements that the sponsor desired that was added to the system. One of the requirements was the constraint to prevent the user to reserve 14 days in advance. The team has
mentioned the discrepancies between the system that was delivered and what the team has promised. The team had high optimistic view and planned to deliver a complete working system. This was not the case, the team was only able to deliver a system that had only partial functionality but was integrated with the database.

**Project Reflection**

What went wrong was that each member did not put the average amount of workload into the project. This resulted in a project that was behind schedule. The project started in the summer, which bring the mentality of procrastination to the mindset to the team members. As Feature Driven Development our methodology, the team was unable to follow all the way through with the methodology. There was no sprints made, the team just simply worked on a feature one at a time. Hopefully, if the team had another chance at this project, the development would be more smooth and the ability to follow the methodology to produce consistent deliverables. However, as the project looked bleak, the team was able to finish the basic functionality of the system. Things that the team would do differently is to motivate themselves to start and work on the project earlier to meet the milestones. What the team learned is the connectivity of the database to the front end design, that MySQL was not a solution for every front end design. The team learned everything that they used

**References**

1. https://docs.mongodb.org/manual/
5. https://angularjs.org/