Basketball Court Rating WebApp for Basketball Enthusiasts

CEN 4010, Software Engineering I

Instructor: Dr. Monique Ross

Group: 2

Members:
Adrian Rodriguez
Oscar Padilla
Joseph Nguyen
Amir Jindani
Alex Campaneria

I. Introduction

The Application and Its Users

This web application is aimed at basketball players, specifically enthusiastic players looking for a basketball court that meets their specific needs. Salient characteristics of the various types of users that would be using this product include traits like being new to an area, planning to travel outside of their place of familiarity, finding a suitable court to perform exercises, looking for a court with specific amenities, looking for a court with specific physical characteristics, or socializing with other players. The two specific characteristics we have chosen to focus on are users who are enthusiastic basketball players with specific needs for a court they with to play on. Our application would aim to solve this problem, while also providing ways to provide feedback to basketball courts.

Differences and Implications

This basketball court rating application allows the community to share their opinions on visited basketball courts, using user input such as star ratings ranging from 1 to 5, and allowing comments for discussion of a court. What differentiates this web application from other similar court rating applications is the ability for users to share more detail and personality in their reviews, with added predefined characteristics to highlight the most crucial court information. This application will allow the basketball community to become more involved which potentially leads to the implication of a growing community, and a potential attraction for local companies to sponsor basketball courts. For example a highly rated or popular court that sees a lot of traffic on the application could attract potential sponsors that could tie their brand to the court, or even use their brand on the application, if advertising support is implemented. An additional implication that the application could have is the possibility of expanding to other sports that use fields or courts, such as tennis or soccer. Note that these implications rely on an active use of the application by a large community, hence a constraint is placed on our application.

Additionally, since the application is so reliant on users, there must be an active community to keep information about the courts in the database accurate. We then lead to the fact that if a community is very active, then there must be moderation performed on the information gathered of a court and the user submitted comments to make sure that users are behaving appropriately. One final constraint is that throughout development there will be no active users of the application, except for the developers, so dummy data will have to be used for testing and demonstrations.

Some negative implications that the application could produce could negatively impact the parks or businesses that own courts. These entities that own these courts could receive reduced traffic if their courts are rated badly, or are if their courts receive negative comments. We also provide statistics on courts, such as frequent visited times

and average age of users who visited a court. This information could be used nefariously by users of the application.

User Stories

#1 Create User Sign Up
As a developer, I need to create a user sign up interface so that a user can create an account.
Acceptance Criteria
Verify that a user can successfully sign up.

#2 Create User Login
As a developer, I need to create a user login interface so that a user can access their account if they have one.
Acceptance Criteria
Verify that a user can successfully log in.

#3 Create User Logout
As a developer, I need to create a log out interface so that a user can sign out of their account.
Acceptance Criteria
Verify that a user can successfully log out.

#4 Display Court Attributes
As a user, I want to be able to see various attributes of a court such as hours open, size, hoop size, membership requirement, indoor/outdoor status, and net status, so I know what to expect when I visit that court.
Acceptance Criteria
Verify that these attributes can be displayed to a user when they view a court, if available.

#5 Store Court Attributes
As a developer, I need to create database attributes on a court table to store various values of court attributes, so that they can be displayed to a user.
Acceptance Criteria
Verify that the database can store the defined values for court qualities.

#6 Define Court Attributes
As a developer, I need to define court attribute descriptors such that they are logical and can be shown to a user without confusion.
Acceptance Criteria
Verify that hours open can be defined in a format of XX:XX[AM,PM], that size can be defined as [Full, Half, Both], that hoop size can be defined as [Short, Medium, Regulation], that membership requirement can be defined as [True, False], that indoor can be defined as [True, False], and that net status can be defined as [True, False].

#7 Display Court Qualities
As a user, I want to be able to see qualities of the court such as cleanliness, pavement condition, and net condition, so I know what to expect when I visit a court.
Acceptance Criteria
Verify that these qualities can be displayed to a user when they view a court, if available.

#8 Store Court Qualities
As a developer, I need to create database attributes on a court table to store various values of court qualities, so that they can be displayed to a user.
Acceptance Criteria
Verify that the database can store the defined values for court qualities.

#9 Define Court Attributes
As a developer, I need to define court quality descriptors such that they are logical and can be shown to a user without confusion.
Acceptance Criteria
Verify that cleanliness, pavement condition, and net condition can be described as [Excellent, Good, Adequate, Terrible].

#10 Display Court Amenities
As a user, I want to be able to see the amenities that a court has to offer, such as water fountains, vending machines, and bathrooms, so I know what to expect when I visit.
Acceptance Criteria
Verify that these amenities can be displayed to a user when they view a court, if available.

#11 Store Court Amenities
As a developer, I need to create database attributes on a court table to store various values of court amenities, so that they can be displayed to a user.
Acceptance Criteria
Verify that the database can store the defined values for court amenities.

#12 Define Court Amenities
As a developer, I need to define court amenities descriptors such that they are logical and can be shown to a user without confusion.
Acceptance Criteria
Verify that water fountain, vending machine, bathroom can be described as [Available, Unavailable].

#13 Submit Court Rating
As a user, I would like to submit a rating of a court, so that I can contribute to the rating of a court.
Acceptance Criteria
Verify that a user can submit a rating for a court.

#14 Display Court Rating
As a user, I would like to see the average rating of a court so that I can know how other users of the application view the court.
Acceptance Criteria
Verify that a user can view the average rating of a court.

#15 Store Court Rating
As a developer, I need to create a rating table to store the rating of a court, ranging from 1 to 5, so it can be aggregated and averaged to get an average rating of a court.
Acceptance Criteria
Verify that the database can store a rating of a court. Verify that the average can be calculated successfully.

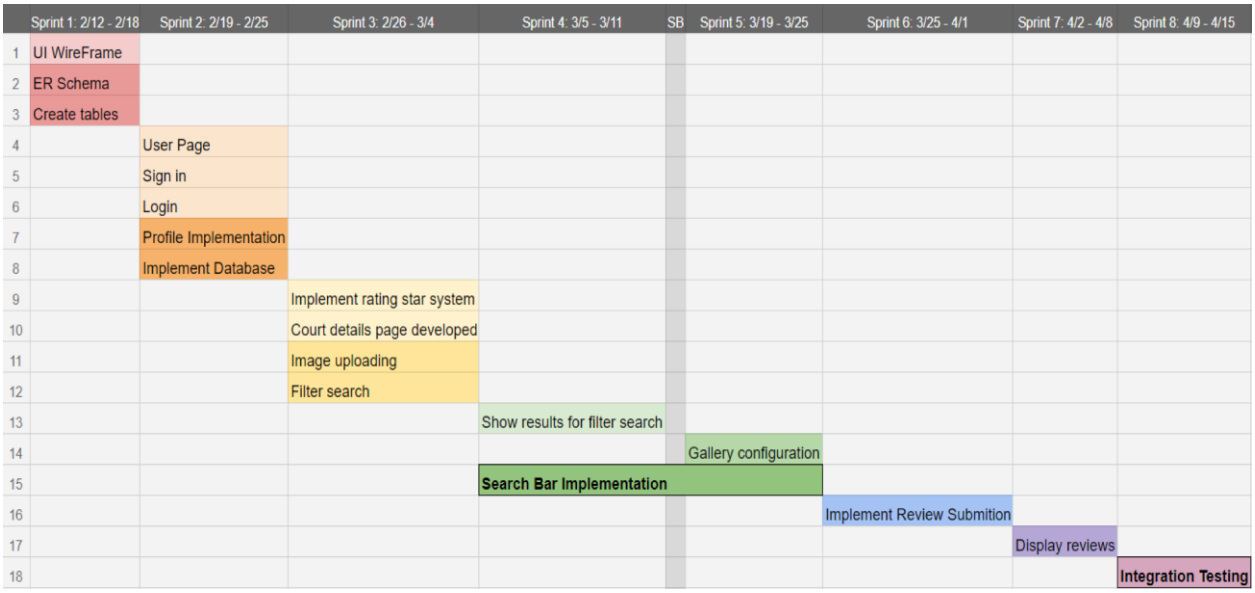
#16 Submit Court Comments
As a user, I want to be able to comment my opinions and experience with a court, so that information can be available for other users.
Acceptance Criteria
Verify that a user can submit a comment

#17 Store Court Comments
As a developer, I need to create a database table, court comments, so that a user can store their comments on a court.
Acceptance Criteria
Verify that the database can store user comments.

#18 Display Court Comments
As a user, I want to be able to see other user’s comments, so that I can make a better decision on whether or not I wish to visit a court.
Acceptance Criteria
Verify that a user can see other user’s comments on a court.

Schedule

To determine the allocation of backlog items to our sprints, we first used the planning poker method on each of the user stories to assign a difficulty from the first few terms of the Fibonacci sequence, which helped calibrate the team's sense of difficulty for development activities. We then used the MoSCoW process to allocate the user stories to the sprints, first by identifying foundational user stories that are necessary for the application to run and adding them to the first few sprints. Allocation of 'stretch goal' user stories were given to the later sprints. Sprints are typically one week in length, consisting of two user stories split between the front-end and back-end developers which the team had determined required a high level of effort to complete.



Roles

Role	Members	Description
Front-end Developer	Amir Jindani Alex Campaneria	Will be focused on designing the layout and user interaction for the application
Back-end Developer	Oscar Padilla Joseph Nguyen	Will be focused on database design, implementation of frameworks, and connecting front-end components to back-end modules.
Full-stack Developer	Adrian Rodriguez	Will be focused on helping both the front and and back end teams with any tasks.

Motivation

The motivation for this basketball court rating application stemmed from a few group members sharing an interest in basketball who thought “what if I could find a basketball court that met my personal standards”. With this interest in basketball and the technical experience brought to the table by Adrian, who has JavaScript work experience, and Amir, who has a strong familiarity with HTML5 and CSS, the team came to unanimous decision that a web application was the best platform for our idea. Along with Alex, currently enrolled in a Human Computer Interaction course, can integrate key concepts of UX into the project. We also recognized that developing a web application would give everyone experience with modern industry technologies, such as Node.js and React, and methodologies used in the workforce today, like Agile development.

II. Design

Defining Design

Designing is the process of defining a series of components for a system so that it satisfies a set of specified requirements. In the case of our project, we used UML rules to model an application use case, class, and sequence diagram to define how a user should interact with the application in a specific scenario.

Design Process

Initially, designing this project was troublesome. The technology stack that we are using is not a stack that we are all familiar with. Therefore, a lot of time was spent by the group looking through the React library framework in order to understand how the library was used to create a program. After gaining some understanding of React’s components, and what was required in order to have a back-end that spoke to a database, we started designing our diagrams. Our previous drafts were created with little to no structure, and were riddled with formatting and syntax errors because of our unstructured approach. After a critique of the diagrams by Danny, the learning assistant, and after a team meeting, we decided to sit down and really look at the bones of our application.

First, we created a use-case diagram that outlined a user and services that would interact with our overall application. We thought we would outline the bare bones use-case by having a user be able to login/signup, search for a court, view a court, rate a court, add a comment to a court and remove their comment to a court. We also had some data verification on those login/signup actions.

From there, we created our class diagrams by trying to stay true to UML rules, while still maintaining the paradigms that React and JavaScript allowed us to use for our application. As you can see in the class diagram, we have 2 main parts of our application, a server and an app. The server is a utility that allows us to enable data functionality to our application, and the application container is what lets a user interact with the server. Our application has multiple components that are rendered upon the URL of a user's browser. The main components are connected to the application. Any sub-component is generated by the component it is attached to. Notice that most of the return types and variable types are objects. This is because JavaScript is a loosely typed language, so the objects simply consist of key-value pairs that vary depending on the object. This goes against UML's rule of needed to explicitly state the type of the variable, but we tried to adhere to the rules nonetheless.

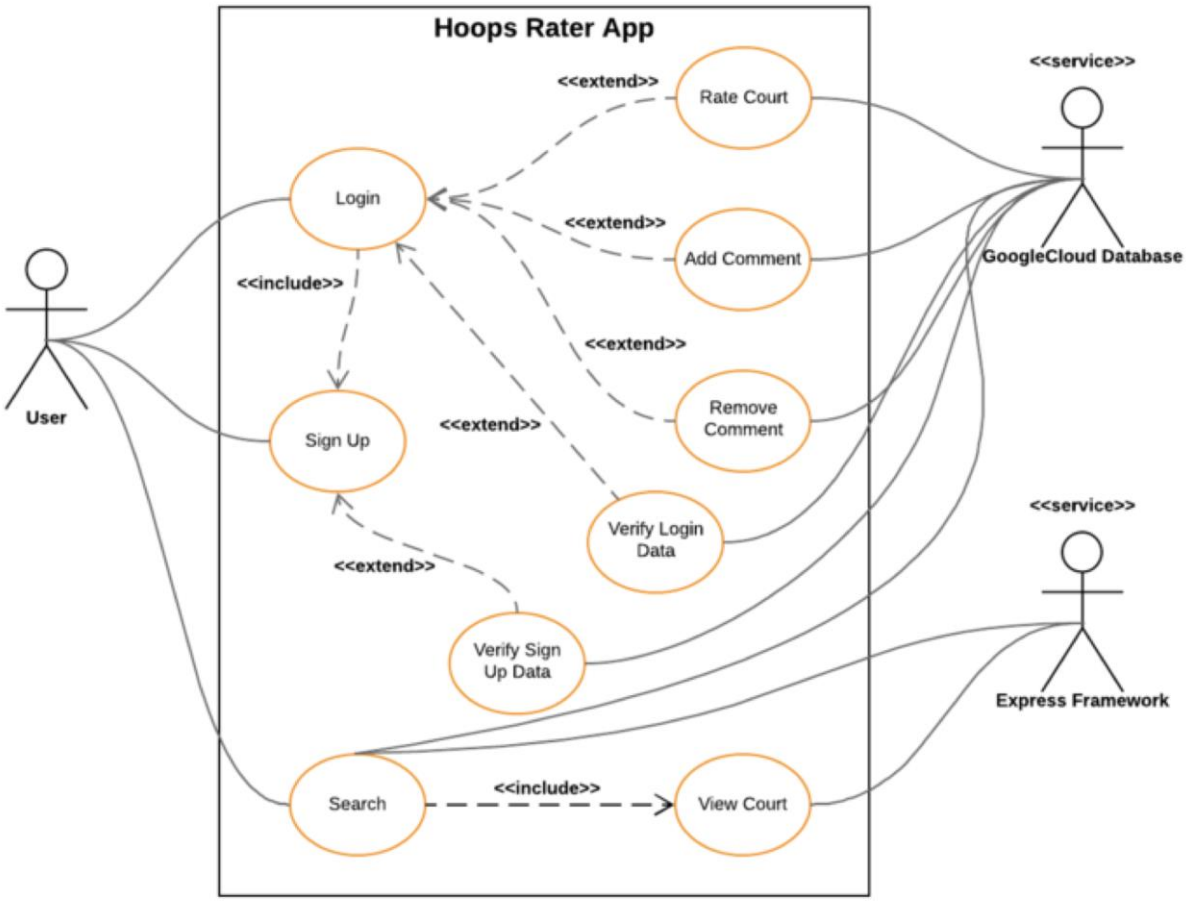
Finally, once we had our use case and class diagrams, we were able to make a sequence diagram of a user logging into our application, as displayed in the diagram section.

Diagrams

In our diagrams, we first design a system-level use case diagram while also making a system-level class diagram. We then create a sequence diagram using the login use case.

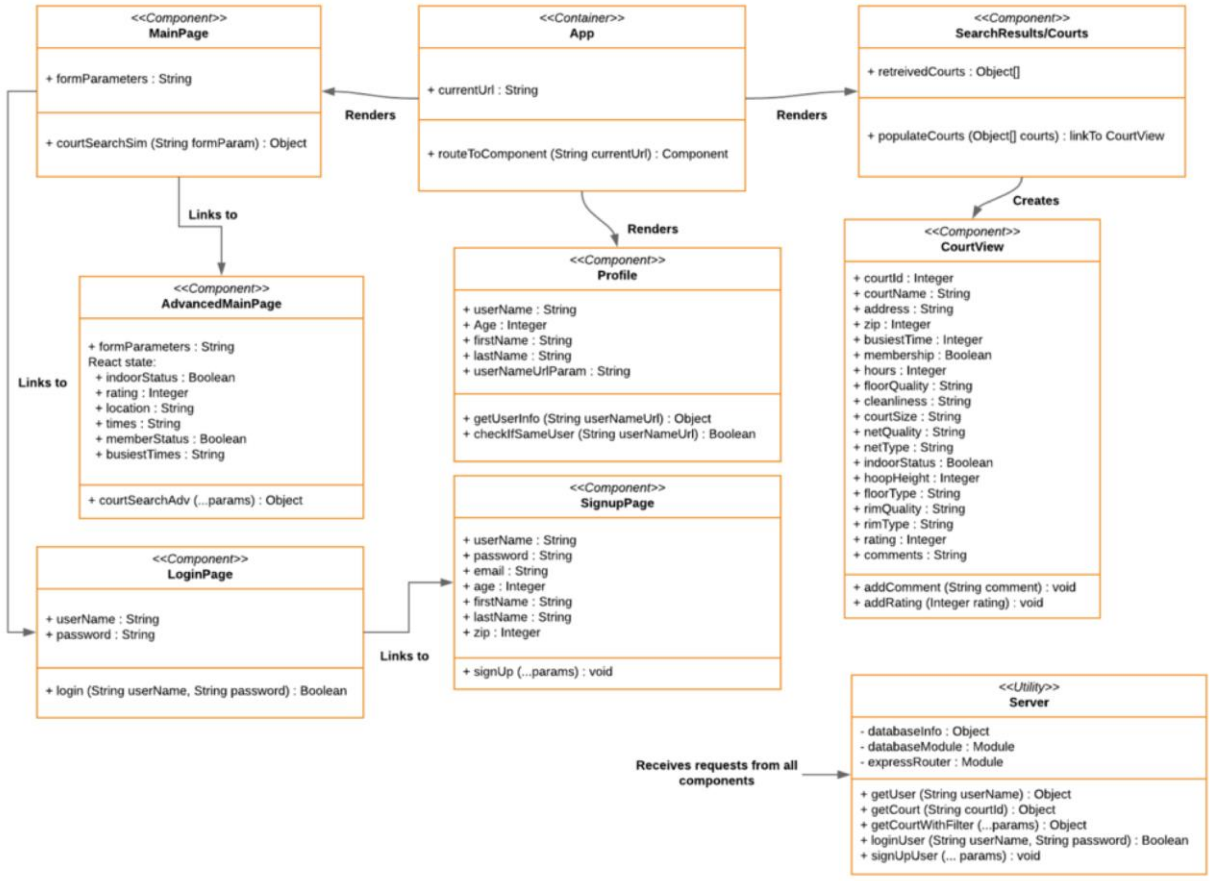
System-Level Use Case Diagram:

Here we demonstrate a system-level use case diagram where a user interacts with our application as a whole.



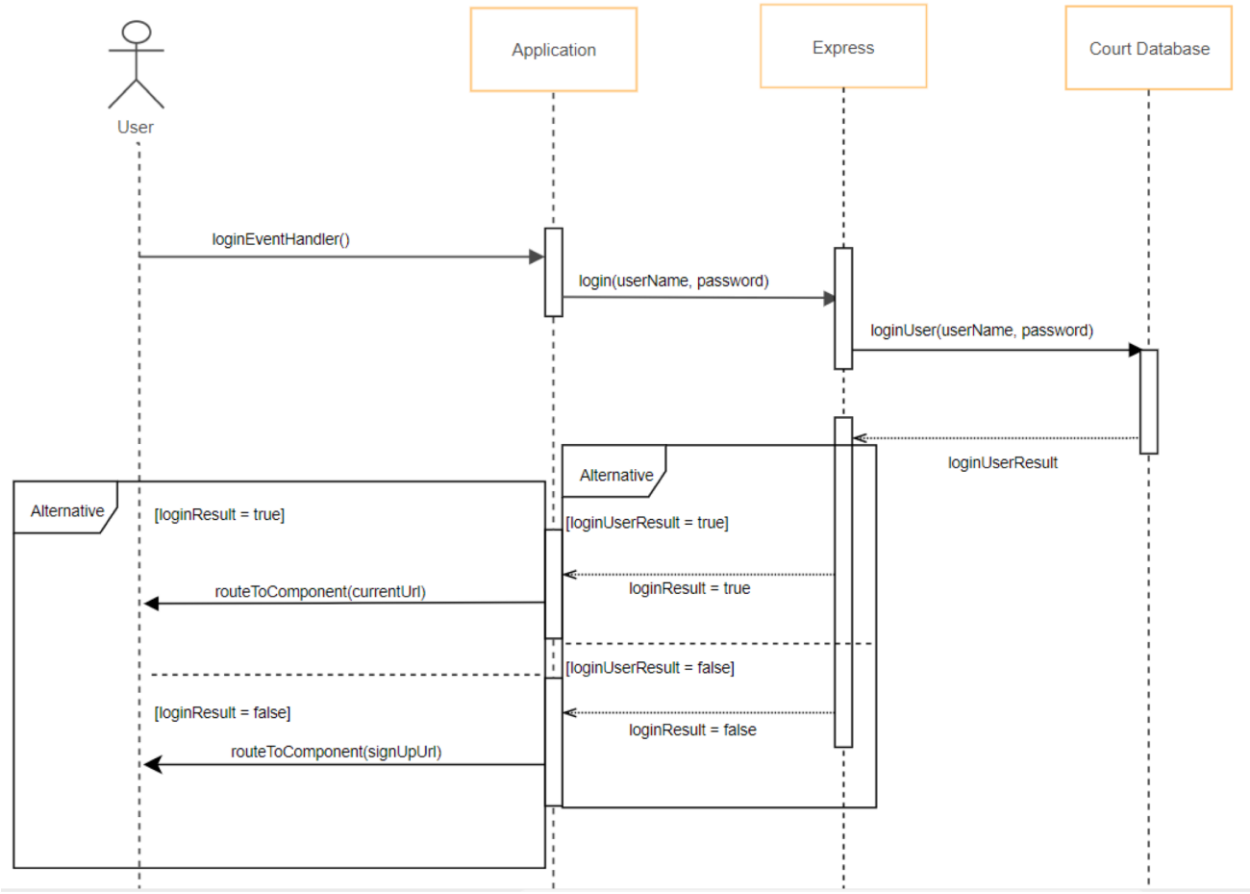
System-Level Class Diagram:

In this diagram, we demonstrate class diagrams for components that our application requires at a system-level.



Login Sequence Diagram:

In this sequence diagram, we demonstrate a user logging into the application, referencing user story #1.



Rationale Management

Issue 1: Platform — Web application vs. mobile application

Once a general idea of the basketball court rating application was brainstormed, we then had to decide on the best platform suitable for our application. We had been considering a platform on either web or mobile that would meet our criteria, the main criteria being usefulness and accessibility. The main benefit of building a mobile app is accessibility since nearly everyone has access to a cell phone. But we ultimately decided to build a web application because it would be easier to develop and it is more universal. Building a phone application would require support for multiple operating systems and

devices, while a web application requires less individualized tweaking yet can still be used from a mobile device.

Issue 2: Database type — Relational database vs non-relational database

Because our project is a user driven court rating application, a database to store user and court information was required. In choosing the most appropriate database type we had to consider the best match with how our data was expected to be handled. The two options considered were a relational database using PostgreSQL and a non-relational database using MongoDB. Due to the wide popularity of MongoDB for web developing, we first began setting up our application around MongoDB but soon enough when it came time to create the database we realized that noSQL and a document-based database was counter-intuitive to the function of our application. We decided to do more research into database designs to fully comprehend our options.

Since our database structure needed strict schemas with defined attributes, such as a court's details and a user's information, it made sense to use a relational database with tables as created with SQL. On the other hand the non-relational noSQL database used with MongoDB meant that the database would have to be document-oriented where there are no tables and each entry into the database is as one whole independent unit with its own individual attributes. Although MongoDB's noSQL document-oriented database management system is more flexible and trending in popularity, we realized after discussion that our application's database had to be much more defined and structured and PostgreSQL was the appropriate relational database management system to use.

Issue 3: Search filter functionality

Searching is an important part of our application. We decided that this functionality will take an address or name of a court as a user input and display the corresponding results as an output. We would have a search bar for the user to enter a name or address of a court they are querying, and if needed, additional filters for narrowed down search results. Making sure that the user would be able to interact with the search in a usable way was key in resolving this issue.

Two options were considered. One implementation with a basic name or address search with an in-line filter feature, or a basic name or address search with a link to an advanced search page where you could apply filters before searching. To resolve this issue, we discussed how an advanced search bar would help our users find their preferred court with ease. We added on to our first option and combined both options to come to a more simplified conclusion.

When making our decision, we thought about what would be the best way for a user to intuitively find a basketball court that would meet their needs, and how the

structure of the search component would add to an easier site-navigating experience. We debated of combining the filter and search functionality, but finally made our decision on a group vote, taking into consideration the details mentioned above.

III. Verification

Defining Verification

Verification is the process of checking that software system meets specifications and satisfies the expected requirements with the use of testing.

Verification Process

For our verification process, our team used quality assurance testing. When our team was ready to push out a major feature (searching, filtering, et cetera), we ran through a user interaction and verified the input and output of that user interaction.

Testcase Demonstration

Our quality assurance test cases were verifying that a user could go through a list of actions and complete a task, which will be described below.

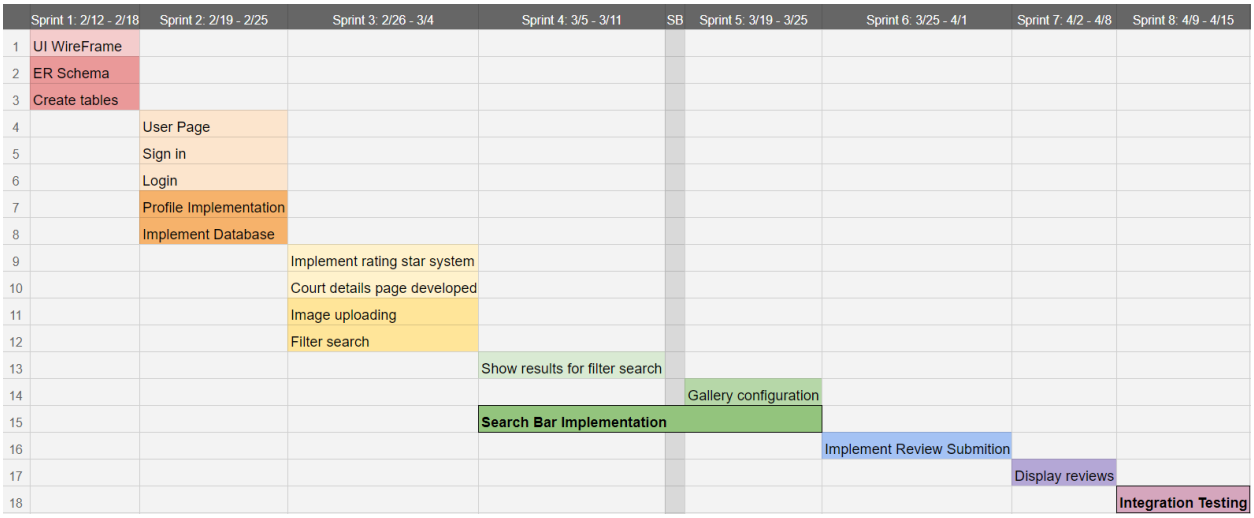
Purpose: Verify that the development environment could be started.		
Step	**Test Step**	**Expected Results**
1	Clone the repository onto the local machine	The clone results in the project being stored on the local machine
2	Maneuver into the directory by using the command ‘cd hoops-rater-app’	Able to successfully access the root directory
3	Run the command ‘yarn’ in the root of the project	Yarn installs all dependencies required at the root level of the project
4	Maneuver into the client directory by using the command ‘cd client’	Able to successfully access the client directory
5	Run the command ‘yarn’ in the client directory	Yarn installs all dependencies required at the client level of the project
7	Start the development environment by using the command ‘yarn dev’	The machines default browser opens to the url localhost:3000

Purpose: Verify that a user can search for a court.		
Step	**Test Step**	**Expected Results**
1	Go to ‘localhost:3000’ in the browser	The web application loads on the main page
2	Click the ‘make a search’ button	The browser loads the search page
3	Enter a basketball court name	Able to type text into the field
4	Press [Enter] or the ‘Search’ button	The browser loads a list of results with matching names
5	Click on the name of a court	The browser loads the court page for that specific court

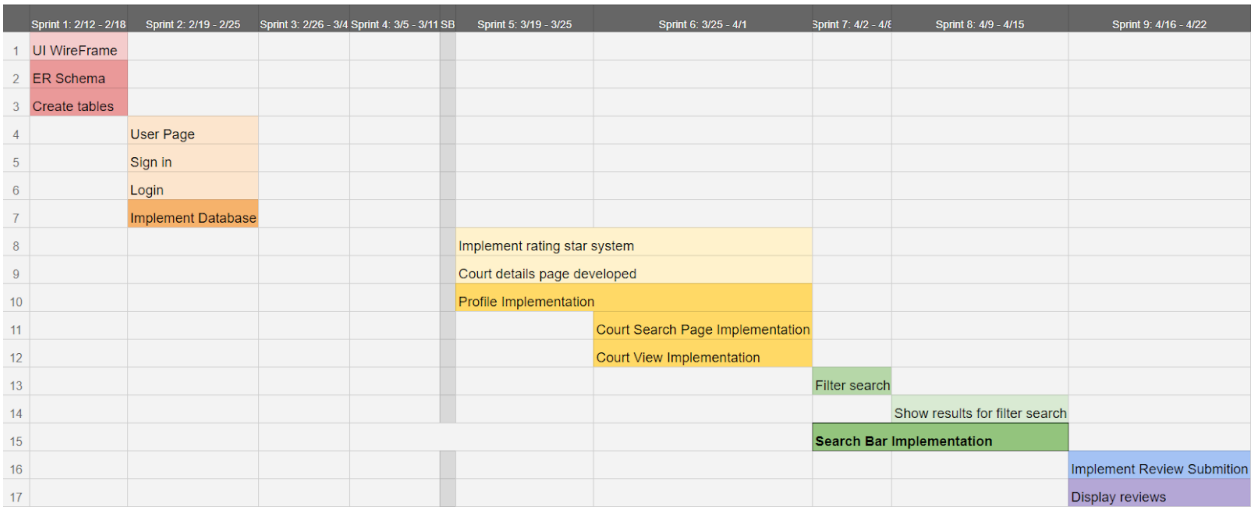
Purpose: Verify that a user can make an advanced search for a court.		
Step	**Test Step**	**Expected Results**
1	Go to ‘localhost:3000’ in the browser	The web application loads on the main page
2	Click the ‘advanced search’ text underneath the search bar	The browser loads the advanced search page
3	Enter a zipcode	Able to type text into the field
4	Select a busiest time	Able to select a value from the dropdown
5	Select whether or not a membership is needed	Able to check the checkbox
6	Select the open time for a court	Able to select a value from the dropdown
7	Select the close time for a court	Able to select a value from the dropdown
8	Select whether or not a court is indoors	Able to check the checkbox
9	Select a rating for a court	Able to select a value from the dropdown
10	Press [Enter] or the ‘Search’ button	The browser loads a list of results matching the options selected
11	Click on the name of a court	The browser loads the court page for that specific court

IV. Reflection

Timeline



PROPOSED SCHEDULE



ACTUAL SCHEDULE

The greatest deviation from the proposed schedule is our 2 week push back of sprints 3 and 4 due to requiring more time for learning more about the technology and for project preparation. Another deviation from the proposed schedule was a one week extension for sprint 3 due to underestimating the time required to implement our application's numerous pages. The last deviation was a re-scoping of our project, where, after reviewing our progress, we decided that due to time constraints image uploading and gallery configuration weren't a 'need' for our application.

Lessons Learned

Flexibility isn't an option but a necessary skill. By the end of the semester, our team learned that when selecting a project idea there are some very important questions that must be asked and answered. What is the purpose and scope of the project? What technologies and internal or external resources will we be needing? We answered these questions a little too vaguely at the beginning of our project, thus our team had to trim down many features we would not be able to implement on time.

We originally planned for 1 week sprints, but this turned out to be very optimistic. We learned that although a lot of planning is put into the Gantt chart, it has to be continuously modified. This occurred when we gained a new understanding of the project after re-scoping. As some tasks got stretched across multiple weeks, the Gantt chart had to be revisited.

We also learned firsthand how much effort goes into planning before actual coding. It's important not to jump into the project too quickly without understanding the purpose of all the selected technologies. UML, specifically use case diagrams, have helped our team have a more tangible idea with a more specific goal. Rather than saying we'll have a backend communicate with our front end, we now can say our backend Express API has to make a query to the database stored on Google Cloud and then send the results to our front end React components. These models also help our idea be more concrete and allow for more efficient communication across team members.

We've found that the best form of communication was always face-to-face, weekly meetings and daily stand-ups. This allows the other team members to have an idea of how far along the application is and what upcoming tasks would be tackled next. Multiple team members worked on the same component, so it was very helpful to communicate about the type of inputs the functions took and the type of output it returned. We could then coordinate to finish the component on time for the sprints.

Overall this process provided ample opportunities for learning and growth that we would not have gained outside this course.