CS482/495/496 Software Project Proposal: add your tentative project title here

Nate Barton, Joe Fielding, Kyle Richards, Vilnis Jatnieks

2025-11-10

1 Client Information

By sharing this client information and the rest of this document, you are stating that this client has provided this project as something they want (not something you created and asked if they wanted), and that they are interested in having you complete this project for your capstone.

- Client name: Dr. Isaacman
- Client title: Department Chair, Associate Professor
- Client email address: snisaacman@loyola.edu
- Client employer: Loyola University Maryland
- How you know the client: CS department

2 Project Description

2.1 Overview

The League of United Minors (LUM) youth football league website will provide a space for community interaction, assist coaches and players in roster management and team sign-up, and display real time data including live games, standings, and league-approved social media posts.

2.2 Key Features

[At this point you should have a basic understanding of your client's needs. List out the key features of the software system the client wants you to build.]

2.3 Why this Project is Interesting

[Why did you decide this project was interesting enough to you to be a capstone project? What about this project is enticing? Why should anyone care?]

2.4 Areas of CS required

[What subfields of computer science seem most likely to be relevant to your project? A capstone must involve multiple.]

2.5 Potential Concerns and Questions

[Is there any aspect of this project that makes you unsure if it will work, either due to your own interests/background, or that you aren't sure if it fits the requirements? Are there questions you have about this project that you want instructor feedback about?]

2.6 Summary of Efforts to Find a Project

(Not necessary for 482) [Briefly list out when/how you've discussed with this client, and if you've discussed with other clients who either didn't work out or didn't respond. If you considered a different project and it didn't work out, why didn't it work out?]

[Most CS495 projects end here. The sections below are for CS482 and CS496 software projects].

2.7 Comparison to Draft

[For CS496 only, focus on highlighting the major differences between the draft proposal in CS495 and this one here. If there are no major differences, you can remove this subsection.]

3 Requirements

3.1 Non-Functional Requirements

[Non-functional requirements are just as important as functional requirements. Dont forget to specify them.]

ID	NFR Title	Category	Description
NFR1	NFR Example 1	Usability	Description of the NFR (it does not follow a user story template)
NFR2	NFR Example 2	Security	Description of the NFR (it does not follow a user story template)

Table 1: Non-Functional requirements

3.2 Functional Requirements (User Stories)

[In CS482, all functional requirements are written as User Stories. In CS496, some projects may use a different template to write the requirements. The table below is an example of writing the Stories. Adapt accordingly to different templates or if you want to display more info.]

ID	Story Title	Points	Description
S1	Story Example 1	5	As a user, I want to write a user story example, so that people will understand them.
S2	Story Example 2	2	As a user, I want to write a user story example, so that people will understand them.

Table 2: Functional requirements as User Stories.

4 System Design

4.1 Architecture

We are using the Web MVC architecture. The main modules for our software are the view, model, and controller. The view will be the several HTML files for each page of our website. When the client goes to the website in their browser, it will send a request to our backend. The controller will be responsible for selecting the appropriate HTML file which will then be sent back to the client. The model will handle all the CRUD operations for our database where we will have tables for all our required entities for our application such as users, teams, and posts. The controller is also responsible for being in the middle of the client's interaction. To show how the three main modules fully interact with each other, here is an example flow of an admin creating a match. The admin interacts with the view on the front end to create a new match. This then

sends a request to the backend, and the controller handles this request. The controller will communicate with the model that the match needs to be added, and the model will add the match to the database.

4.2 Diagrams

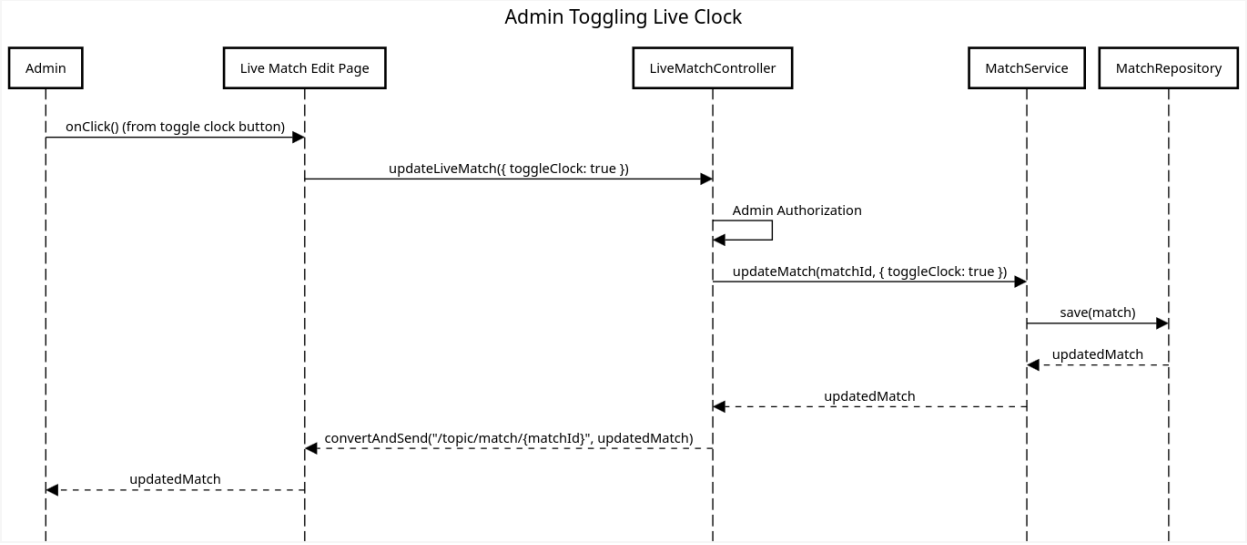


Figure 1: Sequence Diagram showing the steps when the Admin toggles the live match clock

4.3 Technology

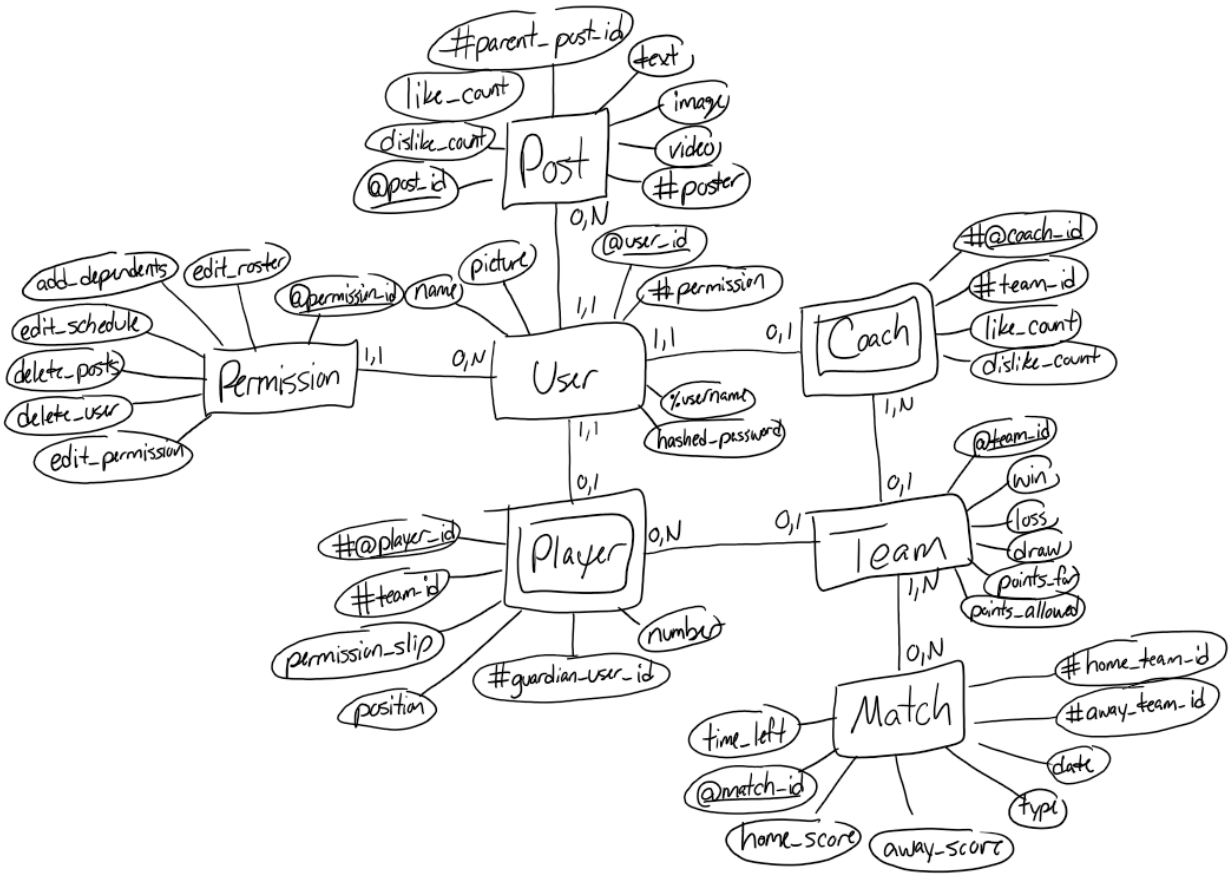
We will be using the Kotlin programming language on the backend. The main framework we will be using to make our web application is Spring. We will also be using JUnit 5 as our unit testing framework for backend, and vitest for frontend testing. We will use React for the frontend, an Mantine for styling and UI components. For the database, we will be using a relational database Postgres SQL.

4.4 Coding Standards

We will use the typical naming conventions in kotlin/java, with class names being Pascal case and variable and method names being camel case. In the database, collection names will be Pascal Case and attribute names snake case. Additionally, only code with at least 80% unit test coverage can be committed.

4.5 Data

4.6 UI Mocks

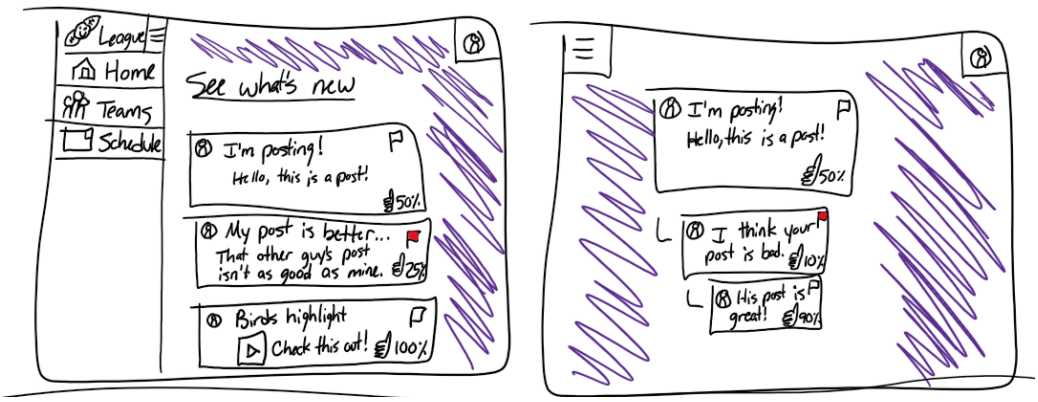


User

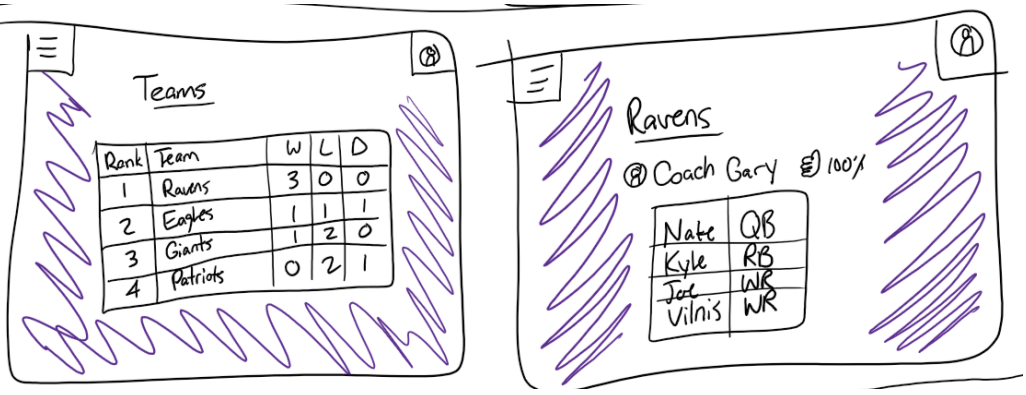
Login/
Profile

This is a minimizeable side bar.

Dashboard / Post

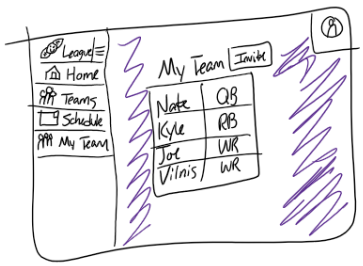


Team view



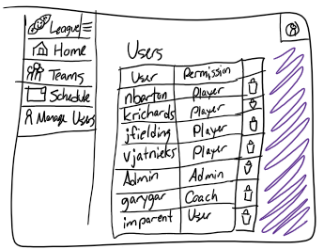
Event view

Coach

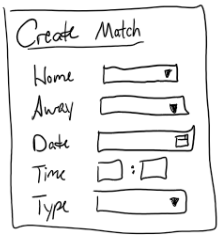


Players also have the "My Team" tab, but are unable to invite players.

Admin



Match creation modal



The admin has "Create" on Schedule page
The admin also has an "Edit" button on Teams page
Also, every comment has 🗑 (delete).

5 Iterations

5.1 Iteration Planning

Iteration	Dates	Stories	Points
1	01/01 - 02/01	A1 Admin manage user, C17 Coach edit roster, A2 Admin create schedule, A25 Admin edit schedule, C3 Coach create team, C16 Coach invite team members, U4 User view team record, G23 Guardian approve child invite	23
2	02/01 - 03/01	A1 Admin manage user, C3 Coach create team, C14 Coach invite players, C15 Coach edit roster, G23 Guardian approve child invite	17.5
3	03/01 - 04/01	S7 Story Title, S8 Story Title, S9 Story Title, S10 Story Title, S11 Story Title	21
4	04/01 - 05/01	S12 Story Title, S13 Story Title, S14 Story Title, S15 Story Title	19
5	05/01 - 06/01	S16 Story Title, S17 Story Title	06
		Total:	70

Table 3: Iteration Planning for Incremental Deliveries

5.2 Iteration/Sprint 1

5.2.1 Planning

We planned to complete:

- coaches create team
- coaches editing the roster
- coach invite team members
- user crud
- admin create schedule
- admin edit schedule
- user view team record
- guardian approve child

Overall this is 23 points
Joe - 6 pts Nate - 6 pts Kyle - 6 pts Vilnis - 5 pts

5.2.2 Work Done

We completed all of the stories that we planned. Each task was delegated as stated in the email detailing who would do each task. The only thing that we did not do is design UI functionality for some of the stories. Despite this, all of the functionality is already there, so we simply need to just add buttons that call that logic.

5.2.3 Testing Coverage

We have high coverage for this iteration and made sure to make unit tests for all the frontend and backend pieces. For the backend, we used Spring's testing framework, which is built off of JUnit. The frontend testing framework we use is vitest. Below are images of the frontend and backend coverage reports at the top level. The full reports are both committed to the GitHub repository.

File		Statements		Branches		Functions		Lines	
components/layout		100%	5/5	100%	0/0	100%	2/2	100%	5/5
components/login		100%	10/10	100%	2/2	100%	2/2	100%	10/10
components/schedule		94.11%	64/68	87.5%	7/8	95.83%	23/24	93.65%	59/63
components/signup		93.75%	15/16	60%	6/10	100%	5/5	93.75%	15/16
components/teams		100%	7/7	100%	0/0	100%	4/4	100%	7/7
request		85.71%	18/21	100%	0/0	71.42%	5/7	85.71%	18/21
types		100%	10/10	75%	3/4	100%	6/6	100%	9/9

Figure 2: Frontend Coverage

Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Branch, %	Line, %
all classes	100% (57/57)	97.3% (180/185)	57.3% (63/110)	95.6% (647/677)

Coverage Breakdown

Package	Class, %	Method, %	Branch, %	Line, %
com.jknv.lum	100% (2/2)	66.7% (2/3)		50% (2/4)
com.jknv.lum.config.seed	100% (7/7)	100% (18/18)	50% (7/14)	100% (95/95)
com.jknv.lum.controller	100% (4/4)	100% (22/22)	50% (1/2)	100% (51/51)
com.jknv.lum.model.dto	100% (10/10)	100% (10/10)		100% (45/45)
com.jknv.lum.model.entity	100% (9/9)	94.3% (33/35)	75% (3/4)	96.7% (145/150)
com.jknv.lum.model.request.account	100% (3/3)	100% (4/4)		100% (18/18)
com.jknv.lum.model.request.match	100% (2/2)	100% (3/3)		100% (20/20)
com.jknv.lum.model.request.player	100% (2/2)	100% (2/2)		100% (4/4)
com.jknv.lum.model.request.team	100% (1/1)	100% (2/2)		100% (13/13)
com.jknv.lum.model.type	100% (3/3)	100% (3/3)		100% (9/9)
com.jknv.lum.security	100% (3/3)	87.5% (14/16)	80% (8/10)	96.5% (55/57)
com.jknv.lum.services	100% (11/11)	100% (67/67)	55% (44/80)	90% (190/211)

Figure 3: Backend Coverage

5.2.4 Retroespective & Reflection

Overall, much of the pitfalls came from the growing pains of learning a new set of technologies. We got the Spring and React frameworks set up. Then we had to figure out the spring and react way of doing things. Though after that, it became much easier to add more. We also did a bit of refactoring, which may have stunted a bit of our progress. This was mainly done because we really wanted to make sure we had a rigid API and proper design, so that in the future iterations we could handle new tasks gracefully. We also perhaps focused too much on security for the login, since we ended up implementing a Jwt system for authentication. Since we have the backbone done and a decent chunk of UI done, we should be able to handle the coming iterations better, so that we can focus more on the features that are necessary. Overall though, we feel that we made decent progress for this iteration, and are ready to make much more for next iteration.

5.3 Iteration/Sprint 2

5.3.1 Planning

We planned to complete:

- A1 admin manage users (2 points) (from iteration 1)
- C3 coach create team (1 point) (from iteration 1)
- C14 coach invite members (1 point) (from iteration 1)

- C15 coach edit roster (1 point) (from iteration 1)
- G23 guardian approve child (.5 points) (from iteration 1)
- A17 provide live feed (3 points)
- U11 user view game (1 point)
- U7 user like/dislike game (1 point)
- U5 user view schedule (2 points)
- G12 guardian approve child account (1 point)
- Spike: refactoring (1 point)
- Spike: live match display (2 points)
- Spike: UI design (1 point)

Overall, this is 17.5 points planned (Nate 4.5, Joe 5, Kyle 4, Vilnis 4)

The idea here was to polish up/finish some of the existing features and then use our existing backend structure for matches and coaches to support live games and coach editing roster

5.3.2 Work Done

We did most of the stories in the list fully, with UI and all. The only story we did not complete is user liking/disliking games (1 point), which was made up for by implementing profile pictures (2 points) along with uploading images/videos to the database (spike, 2 points). So we only deviated from the plan slightly.

5.3.3 Testing Coverage

We achieved a high level of coverage on the backend and frontend, so we think it is a good plan. We are thinking of implementing a code freeze before the deadline as often times we have had to change unit tests due to refactoring or adding new features.

All files

88.36% Statements 562/636 **73.79%** Branches 169/229 **87.01%** Functions 181/208 **90.42%** Lines 529/585

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
components/admin		100%	8/8	100%	4/4	100%	4/4	100%	7/7
components/layout		80%	12/15	60%	6/10	80%	4/5	80%	12/15
components/live_match		86.79%	92/106	67.56%	25/37	85%	34/40	88.29%	83/94
components/login		100%	8/8	100%	0/0	100%	3/3	100%	8/8
components/profile		76.81%	106/138	66.99%	69/103	83.72%	36/43	83.6%	102/122
components/schedule		93.58%	73/78	100%	8/8	92.85%	26/28	93.15%	68/73
components/signup		100%	20/20	94.44%	17/18	100%	8/8	100%	18/18
components/teams		80.82%	59/73	80%	20/25	65.51%	19/29	81.25%	52/64
hooks		95.23%	20/21	83.33%	5/6	100%	3/3	100%	19/19
request		96.94%	127/131	75%	3/4	100%	27/27	96.94%	127/131
types		97.36%	37/38	85.71%	12/14	94.44%	17/18	97.05%	33/34

Figure 4: Frontend Coverage

Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Branch, %	Line, %
all classes	98.4% (62/63)	94.7% (230/243)	62.2% (102/164)	96% (840/875)

Coverage Breakdown

Package	Class, %	Method, %	Branch, %	Line, %
com.jknv.lum	100% (2/2)	66.7% (2/3)		50% (2/4)
com.jknv.lum.config	100% (1/1)	100% (3/3)		100% (4/4)
com.jknv.lum.config.seed	100% (7/7)	100% (26/26)	50% (7/14)	100% (136/136)
com.jknv.lum.controller	100% (6/6)	100% (36/36)	50% (2/4)	98.9% (88/89)
com.jknv.lum.model.dto	100% (9/9)	100% (9/9)		100% (47/47)
com.jknv.lum.model.entity	100% (10/10)	89.2% (33/37)	81.2% (13/16)	97.8% (174/178)
com.jknv.lum.model.request.account	100% (3/3)	100% (4/4)		100% (19/19)
com.jknv.lum.model.request.match	100% (2/2)	100% (3/3)		100% (25/25)
com.jknv.lum.model.request.player	100% (3/3)	100% (3/3)		100% (5/5)
com.jknv.lum.model.request.team	0% (0/1)	0% (0/2)		0% (0/4)
com.jknv.lum.model.type	100% (4/4)	100% (4/4)		100% (12/12)
com.jknv.lum.security	100% (3/3)	77.8% (14/18)	78.6% (11/14)	93.7% (59/63)
com.jknv.lum.services	100% (12/12)	97.9% (93/95)	59.5% (69/116)	93.1% (269/289)

Figure 5: Backend Coverage

5.3.4 Retroespective & Reflection

The most challenging part of this iteration was definitely the fact that tests would break when updating code. This led us to wait until the end to do all of our tests, which took a long time. This is why likes and dislikes weren't implemented, since we needed to focus on testing and couldn't add any new features with the risk of breaking things. For next iteration, we plan to implement a code freeze for our process. This will be a hard deadline for features where afterwards only bug fixes and tests can go through. This will have us not trying to do testing and new features at the same time. Besides that though, implementing the features went decently smoothly as expected. We ended up using websockets for the live matches so that they are actually live. The profile page/profile pictures were a bit of a nightmare, so perhaps we spent a bit too much time on that. Ultimately, we feel that we made great progress this iteration, and we hope to do even better next iteration.

5.4 Iteration/Sprint 3

5.4.1 Planning

[Which stories did you plan for this iteration/sprint. Add the total points for this plan. You can also explain the reason behind your planning, and what major feature(s) your team is focusing on delivering by completing these stories. You may use a table for a summary display of the planning, but elaborate in text more detail in your focus and feature plan.]

5.4.2 Work Done

[Which stories did you complete in this iteration/sprint. Which ones did you partially complete? Who worked on which story? You may elaborate in paragraph(s) to add more detail about the work done.]

5.4.3 Testing Coverage

[Testing is very important. Show your coverage here. Is this coverage good enough? Explain why you think so. Is it not good enough? Explain a plan to increase the coverage. You may also elaborate on why some artifacts do not undergo much testing. If the testing changed from the last iteration, explain the reasons.]

5.4.4 Retroespective & Reflection

[What were the pitfalls, challenges, and issues you had in this iteration? How can you address them to improve the process in the next iteration? Did anything not go according to plan? Why so and how to avoid the same mistake? Write a personal reflection on what you learned in this iteration (even if a small technical thing like Database storage).]

5.5 Iteration/Sprint 4

[CS496 has 5 sprints. CS482 only has only 3 sprints (remove Iterations 4 and 5 from this doc if you are writing a doc for 482]

5.5.1 Planning

[Which stories did you plan for this iteration/sprint. Add the total points for this plan. You can also explain the reason behind your planning, and what major feature(s) your team is focusing on delivering by completing these stories. You may use a table for a summary display of the planning, but elaborate in text more detail in your focus and feature plan.]

5.5.2 Work Done

[Which stories did you complete in this iteration/sprint. Which ones did you partially complete? Who worked on which story? You may elaborate in paragraph(s) to add more detail about the work done.]

5.5.3 Testing Coverage

[Testing is very important. Show your coverage here. Is this coverage good enough? Explain why you think so. Is it not good enough? Explain a plan to increase the coverage. You may also elaborate on why some artifacts do not undergo much testing. If the testing changed from the last iteration, explain the reasons.]

5.5.4 Retroespective & Reflection

[What were the pitfalls, challenges, and issues you had in this iteration? How can you address them to improve the process in the next iteration? Did anything not go according to plan? Why so and how to avoid the same mistake? Write a personal reflection on what you learned in this iteration (even if a small technical thing like Database storage).]

5.6 Iteration/Sprint 5

5.6.1 Planning

[Which stories did you plan for this iteration/sprint. Add the total points for this plan. You can also explain the reason behind your planning, and what major feature(s) your team is focusing on delivering by completing these stories. You may use a table for a summary display of the planning, but elaborate in text more detail in your focus and feature plan.]

5.6.2 Work Done

[Which stories did you complete in this iteration/sprint. Which ones did you partially complete? Who worked on which story? You may elaborate in paragraph(s) to add more detail about the work done.]

5.6.3 Testing Coverage

[Testing is very important. Show your coverage here. Is this coverage good enough? Explain why you think so. Is it not good enough? Explain a plan to increase the coverage. You may also elaborate on why some artifacts do not undergo much testing. If the testing changed from the last iteration, explain the reasons.]

5.6.4 Retroespective & Reflection

[What were the pitfalls, challenges, and issues you had in this iteration? How can you address them to improve the process in the next iteration? Did anything not go according to plan? Why so and how to avoid the same mistake? Write a personal reflection on what you learned in this iteration (even if a small technical thing like Database storage).]

6 Final Remarks

6.1 Overall Progress

[Have you completed everything? If so, present evidence on how you brought value to your client, and the overall client satisfaction. Otherwise, estimate how much progress you done and how long it would take to finish this project.]

6.2 Project Reflection

[Your personal reflection on the project. What lessons did you learned. What would you have done differently. How can you do better work in future projects? You may write this as a team or per person (or both)]

Appendix

[Appendix section if needed]