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

Nate Barton, Joe Fielding, Kyle Richards, Vilnis Jatnieks

2025-11-24

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

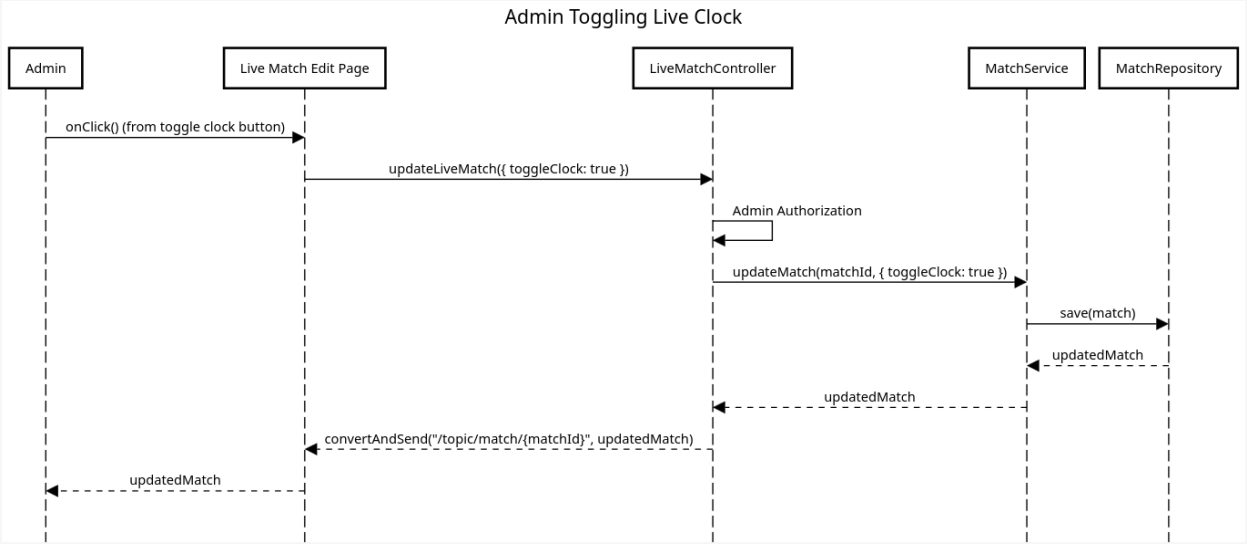


Figure 1: Sequence Diagram showing the steps when the Admin toggles the live match clock

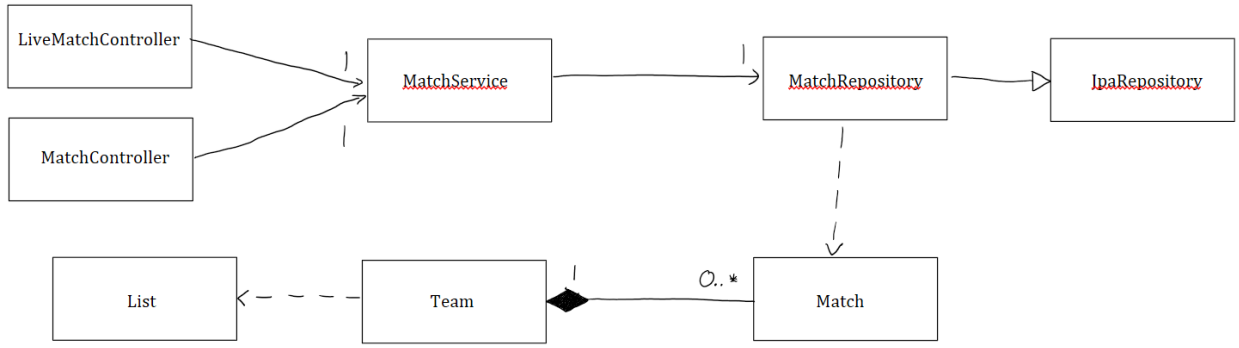


Figure 2: Simplified Class Diagram showing the classes used during a live match

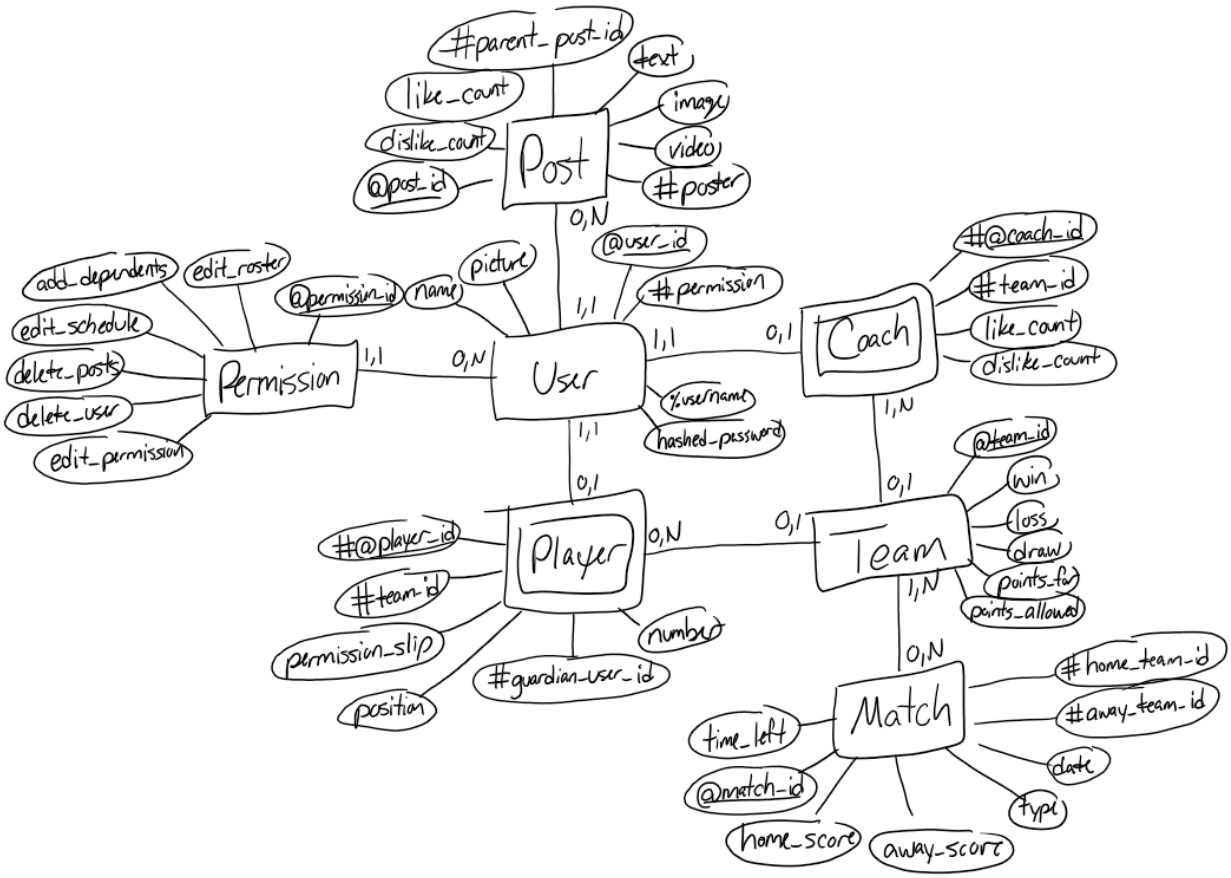
4.3 Technology

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



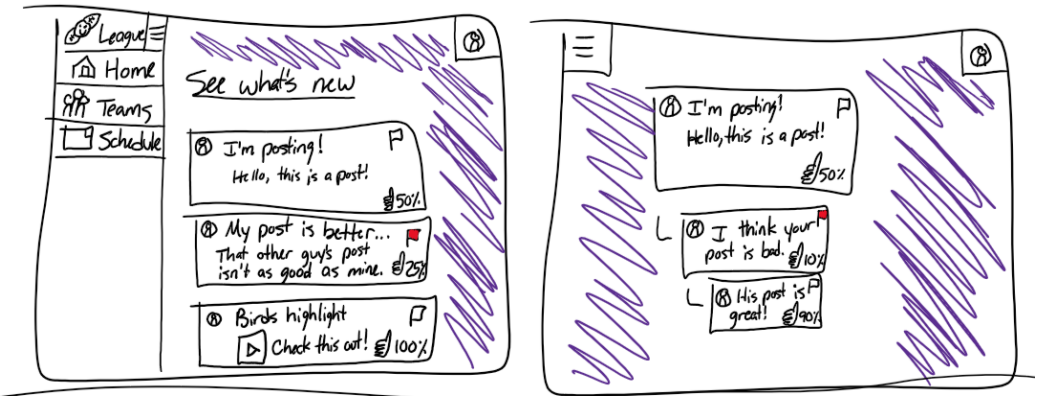
4.6 UI Mocks

User

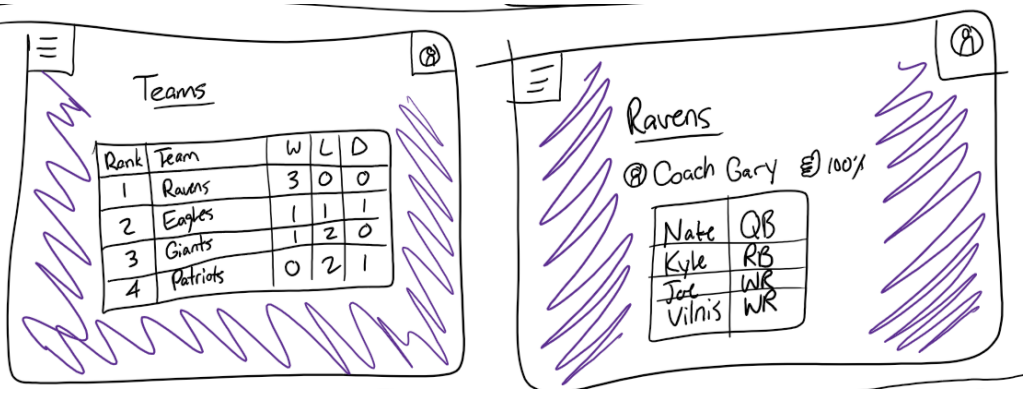
Login/
Profile

This is a minimizeable side bar.

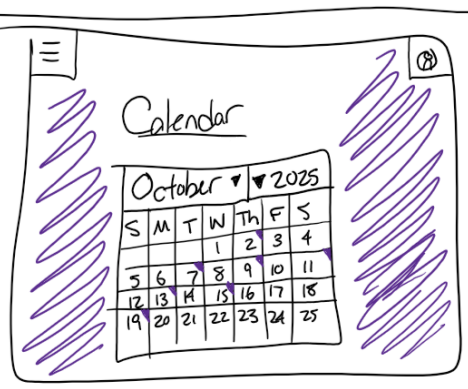
Dashboard/
Post

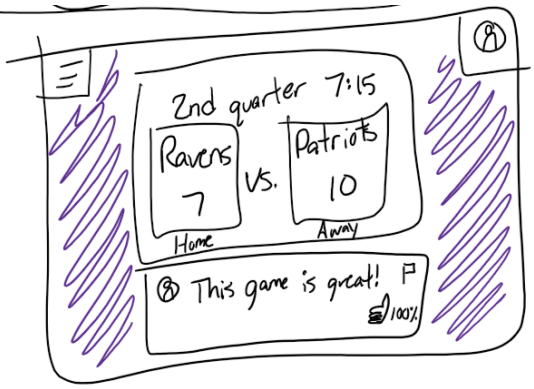


Team view

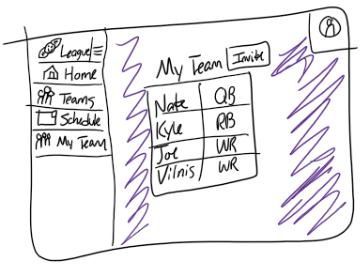


Event view



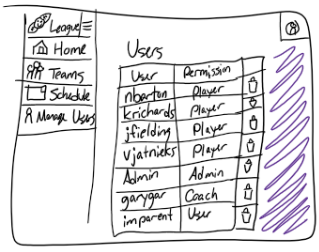


Coach

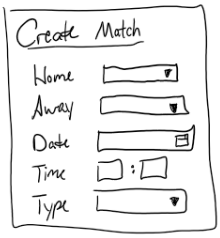


Players also have the "My Team" tab, but are unable to invite players.

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:

Team Member	Story	Story Description	Points
Joe	C17	Coach edit roster (Pair Programming)	1
	A2	Admin create schedule	3
	A25	Admin edit schedule	2
		Joe Total	**6**
Nate	C17	Coach edit roster (Pair Programming)	2
	C3	Coach create team	2
	C16	Coach invite team members	2
		Nate Total	**6**
Kyle	A1	Admin manage user (Pair Programming)	4
	U4	User view team record	2
		Kyle Total	**6**
Vilnis	A1	Admin manage user (Pair Programming)	4
	G23	Guardian approve child invite	1
		Vilnis Total	**5**
		Total	**23**

Table 4: Iteration 1 Planning

5.2.2 Work Done

We Completed:

Team Member	Story	Story Description	Points
Joe	A2	Admin create schedule	3
	A25	Admin edit schedule	2
		Joe Total	**5**
Nate	C17	Coach edit roster (Pair Programming)	1
	C3	Coach create team	1
	C16	Coach invite team members	1
		Nate Total	**3**
Kyle	A1	Admin manage user (Pair Programming)	3
	U4	User view team record	2
		Kyle Total	**5**
Vilnis	A1	Admin manage user (Pair Programming)	3
	G23	Guardian approve child invite	1
		Vilnis Total	**4**
		Total	**17**

Table 5: Iteration 1 Work Done

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 3: 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 4: 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

Team Member	Story	Story Description	Points
Nate	Extra	Refactoring	1
	C3	Coach create team	1
	C14	Coach invite members	1
	C15	Coach edit roster	1
	G23	Guardian approve child	0.5
		Nate Total	**4.5**
Joe	Extra	Live match display	2
	A17	Provide live feed	3
		Joe Total	**5**
Kyle	Extra	UI design	1
	A1	Admin manage users (unfinished)	1
	U11	User view game	1
	U7	User like/dislike game	1
		Kyle Total	**4**
Vilnis	A1	Admin manage users (unfinished)	1
	U5	User view schedule	2
	G12	Guardian approve child account	1
		Vilnis Total	**4**
		Total	**17.5**

Table 6: Iteration 2 Planning

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.

Team Member	Story	Story Description	Points
Nate	Extra	Refactoring	1
	C3	Coach create team	1
	C14	Coach invite members	1
	C15	Coach edit roster	1
	G23	Guardian approve child	0.5
		Nate Total	**4.5**
Joe	Extra	Live match display	2
	A17	Provide live feed	3
		Joe Total	**5**
Kyle	Extra	UI design	1
	A1	Admin manage users (unfinished)	1
	U11	User view game	1
	U7	Profile Pictures	2
		Kyle Total	**5**
Vilnis	A1	Admin manage users (unfinished)	1
	U5	User view schedule	2
	G12	Guardian approve child account	1
		Vilnis Total	**4**
		Total	**18.5**

Table 7: Iteration 2 Work Done

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.

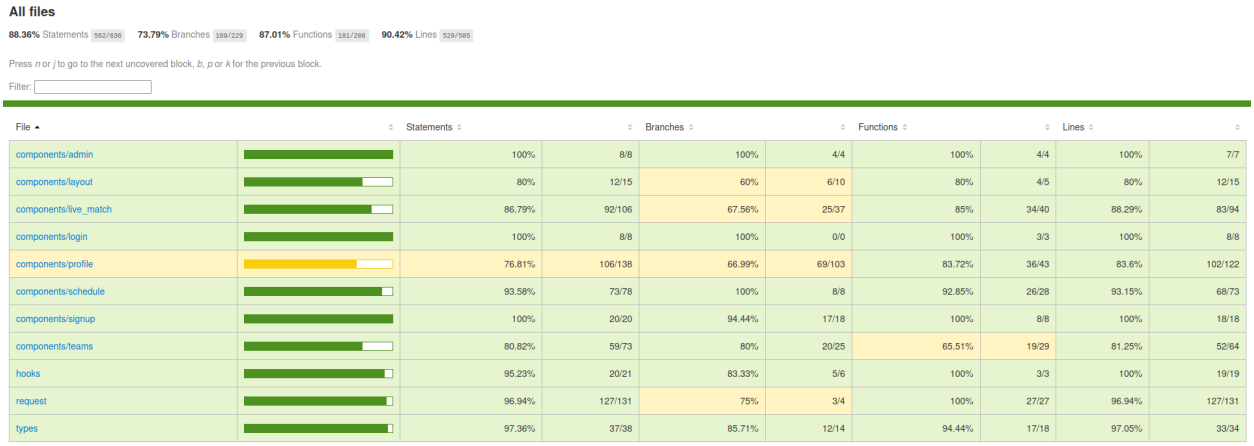


Figure 5: Frontend Coverage

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 6: 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

We planned to complete:

Team Member	Story	Story Description	Points
Nate	A9	Admin review posts	2
	A22	Admin see flagged posts	1
	A24	Admin approve profile picture	2
		Nate Total	**5**
Joe	U8	User create posts (including picture/video)	5
		Joe Total	**5**
Kyle	U6	User comment on games	3
	U7	User like/dislike games	1
	U10	User flag posts	1
		Kyle Total	**5**
Vilnis	A19	Admin upload matches via file	5
		Vilnis Total	**5**
All		**Total**	**20**

Table 8: Iteration 3 Planning

The main focus for this iteration is the posts, as creating posts with images and videos is one things, btu there are are the flags and likes for example which are also related to posts.

5.4.2 Work Done

We finished all the tasks that were planned for this iteration, plus 2 more.

Team Member	Story	Story Description	Points
Nate	A9	Admin review posts	2
	A22	Admin see flagged posts	1
	A24	Admin approve profile picture	2
		Nate Total	**5**
Joe	U8	User create posts (including picture/video)	5
	U18	See playoff bracket	1
		Joe Total	**6**
Kyle	U6	User comment on games	3
	U7	User like/dislike games	1
	U10	User flag posts	1
	G13	Guardian post approval	1
		Kyle Total	**6**
Vilnis	A19	Admin upload matches via file	5
		Vilnis Total	**5**
All		**Total**	**22**

Table 9: Iteration 3 Work Done

5.4.3 Testing Coverage

We achieved good coverage during this iteration on the both the frontend and backend.

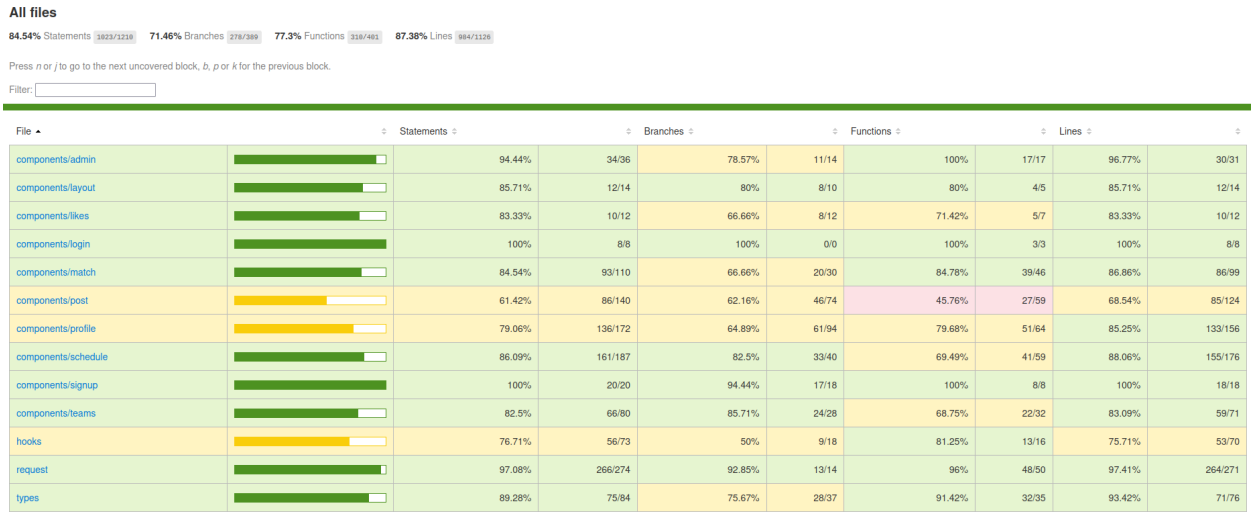
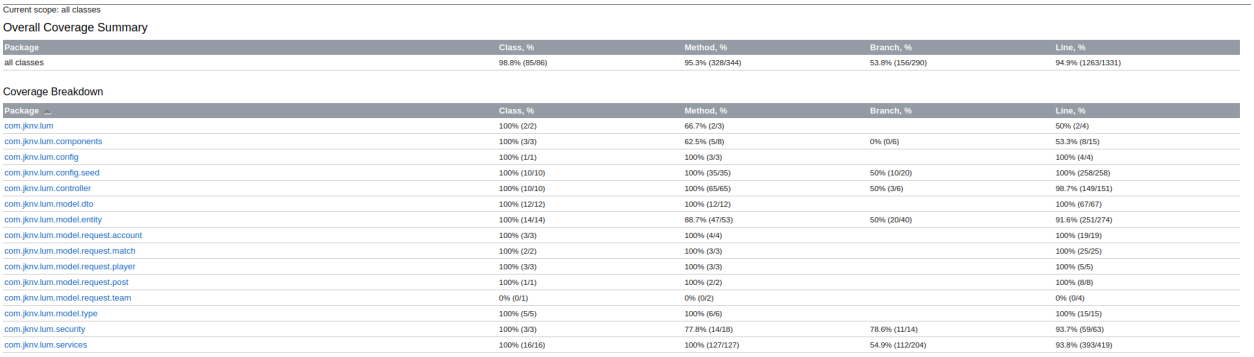


Figure 7: Frontend Coverage



Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Branch, %	Line, %
all classes	98.8% (85/86)	95.3% (328/344)	53.8% (156/290)	94.9% (1263/1331)

Coverage Breakdown

Package	Class, %	Method, %	Branch, %	Line, %
com.jknv.lum	100% (2/2)	66.7% (2/3)		50% (2/4)
com.jknv.lum.components	100% (3/3)	62.5% (5/8)	0% (0/6)	53.3% (8/15)
com.jknv.lum.config	100% (1/1)	100% (3/3)		100% (4/4)
com.jknv.lum.config.seed	100% (10/10)	100% (35/35)	50% (10/20)	100% (258/258)
com.jknv.lum.controller	100% (10/10)	100% (65/65)	50% (3/6)	98.7% (149/151)
com.jknv.lum.model.dto	100% (12/12)	100% (12/12)		100% (67/67)
com.jknv.lum.model.entity	100% (14/14)	88.7% (47/53)	50% (20/40)	91.6% (251/274)
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.post	100% (1/1)	100% (2/2)		100% (8/8)
com.jknv.lum.model.request.team	0% (0/1)	0% (0/2)		0% (0/4)
com.jknv.lum.model.type	100% (5/5)	100% (6/6)		100% (15/15)
com.jknv.lum.security	100% (3/3)	77.8% (14/18)	78.6% (11/14)	93.7% (59/63)
com.jknv.lum.services	100% (16/16)	100% (127/127)	54.9% (112/204)	93.8% (393/419)

Figure 8: Backend Coverage

5.4.4 Retroespective & Reflection

The biggest challenge was testing and introducing new features that causes existing working features to break. For example, we decided to make Matches posts because they are meant to show up on the feedpage, which we did this by making a one to one relationship between these 2. The Post has the foreign key because it is optional as not every Post is a Match. However, this caused Match deletion to break initially because when we tried to delete Matches, it would not work due to the Post having a foreign referencing it. This case was an easy fix as we just also delete the Post in this case, but there were several similar situations to this that happened which we had to be careful about and go back and fix. Additionally, unit testing was a struggle, especially because of refactoring as we not inly had to make unit tests for the new features but also change some of the old ones. This took a tremendous amount of time overall. Though we are glad that we finished the code earlier this iteration, so the last 2-3 days we could focus on testing and fixing a few small bugs. We had said last iteration we wanted to make a code freeze which we did not make an exact one as we pretty much just finished up all the major features before the weekend anyways.

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]