UNIVERSITY OF STAVANGER, NORWAY

DAT240 Project Report (Group 10): Two player image guessing game

Grégoire Guillien, Withya Wijeyaraj, Eirik Ødegård,
Markus Aarekol Johannessen, Alan Shakhewan Rostem and Jari Kunnas

November 4, 2019

Abstract

The purpose of this game is for a guesser to guess the label of an image with as few segments of the specific picture available as possible. There is a proposer player that will select images for the guesser player to see. The game starts with the guesser being shown one segment by the proposer. The objective for the paired players is to succeed with the lowest amount of segments as possible.

This is a web application, where you start by logging in. A logged in user is given the option of playing as a guesser or proposer. Once two players of different roles(guesser and proposer) are ready they will be directed to their own individual views. The proposer will start the game by selecting an image for the game and then the first segment is presented to the guesser.

The Guesser have 3 tries per segment to guess the label. After 3 failed attempts the guesser will be shown a new segment selected by the proposer. The guesser can also ask for a new segment without doing the 3 attempts to guess on the available segments.

The game continues until the guesser guesses the correct label for a given image or when all the segments are shown to the user.

CONTENTS

1. INTRODUCTION

Person in Charge; all

Recent advances in computational capabilities of machines alongside advances in algorithmic intelligence, have surpassed expectations and resulted in staggering feats such as 'AlphaGo' defeating a world champion in the game of Go using deep neural networks. With all the perceived superiority of machines in decision making using Deep Learning we are interested in the question – Do machines think like humans?.

We will attempt to do this by making a two-player game on recognizing images. Two players are paired in the beginning of the game – The Proposer and the Guesser. Such games are called games with a purpose (GWAP).

Rules of the game

The proposer gets to choose which image to use for the specific instance of the game, and has the entire image available during the entity of the game. The proposer then chooses which segments to present for the guesser by clicking on their unique id. The segment that was clicked are then sent to the guesser and now his job is to try to figure out how the entire picture looks like, and the label for it. The proposers job will be to choose the most describing segments in order to make the job easier for the guesser.

Time spent as well as the amount of segments shown are of the essence when it comes to the players final score. The sooner they succeed the better, and the less segments shown for the guesser the better

2. DESIGN

Person in Charge; all

General information on the design is included in this section. More detailed explanation of the detailed design of the game can be found in the designated back-end and front-end sections. There is also an appendix section in the very back of our report that illustrates how we initially visualised the design of our application.

2.1. DESIGN PRINCIPLES

The game was coded with conventional design principles. Classes were encapsulated. There is no implementation of Inheritance or Polymorphism in this project. The game itself is not complicated enough that those designs were deemed necessary. It was more natural to use composition in this project compared to inheritance. An example of this is the composite Party class that uses GameLogic and Player class objects to create a more complex class object using information and functionality from other classes instead of having everything inside itself.

2.2. ACTIVITY DIAGRAM

The activity diagram fig 2.1 explains how the movement for a user in the application would take place. It shows the flow starting from when the user enters the welcome page. Goes through the various guesser or proposer views. Until eventually the game ends and the user can choose to return to the welcome page where they can queue up to play again.

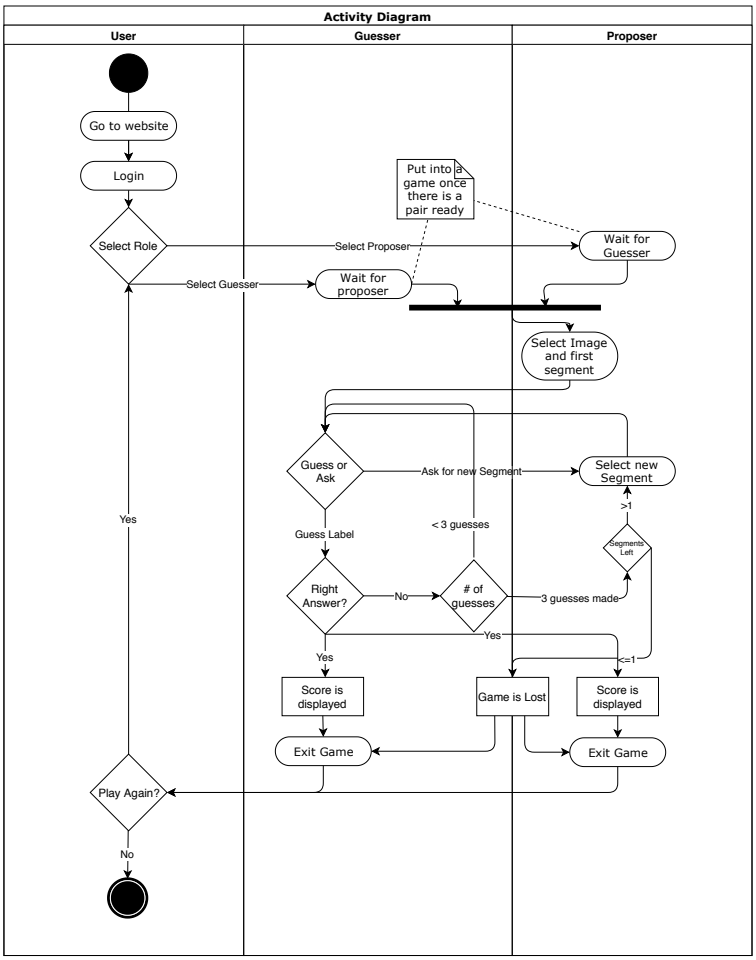


Figure 2.1: Activity Diagram Game

2.3. File Structure

In the figure 2.2 below is a demonstration of how the file structure of the project looks like. Gradle compiles the src/main/java automatically. The rest of the file structure was designed by the group

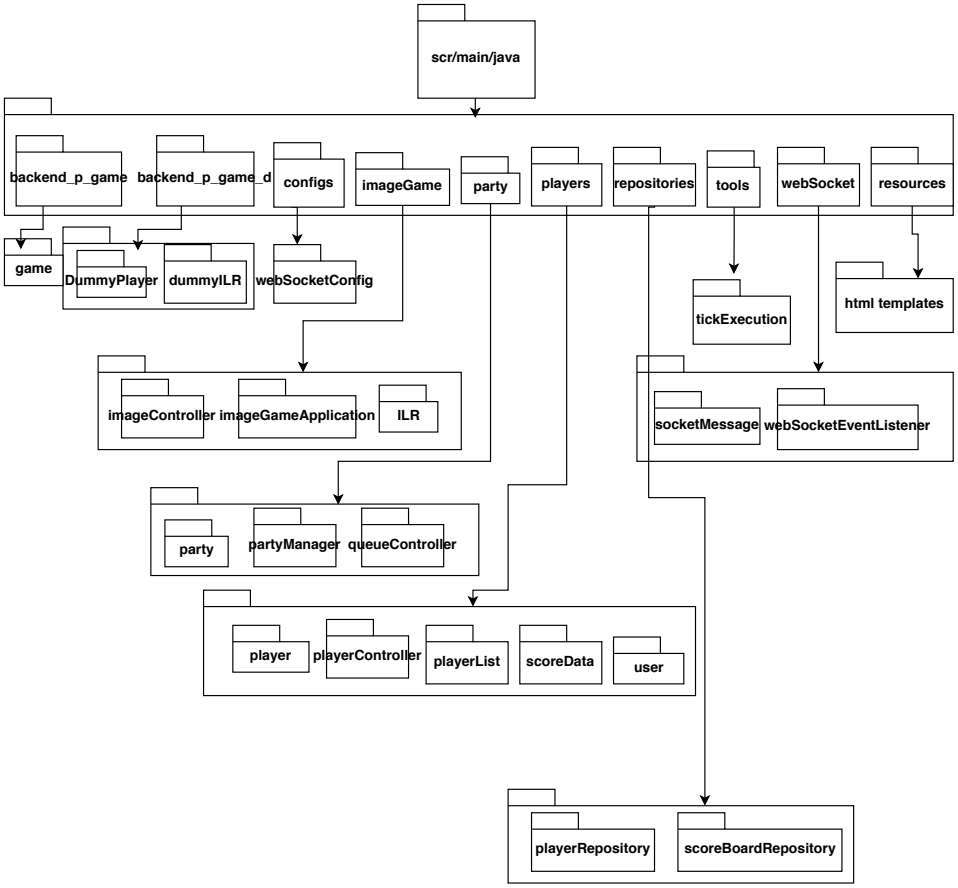


Figure 2.2: Files Architecture of our project

The packages back-end-pseudo-game and the back-end-pseudo-game-dummy was made to enable us to create the structured code for the game in the back-end. By doing this we ensured that the creation of the initial game logic would have no immediate effect on the main application.
Dummy classes were made in the package, back-end-pseudogame-dummy, to simulate players in the game or more if needed be.
We structured the rest of the classes needed for the application in respective packages to make it more readable. i.g in the player-, and the image Game-packages, where we've respectively stored all the classes needed to define players within the application and the classes required to launch the main application.

2.4. TESTING

During the development of this project we tried to have testing in mind while developing. Due to the nature of our back-end architecture it proved difficult to write junit test-cases for the models as they were not simply retrieving objects, but taking in random parameters such as the party id.

Hence, we adopted a user driven testing methodology, where we added print statements for being updated when functions were run and updated.

3. BACKEND

Persons in Charge; Alan, Eirik and Markus

3.1. BACK-END SUMMARY

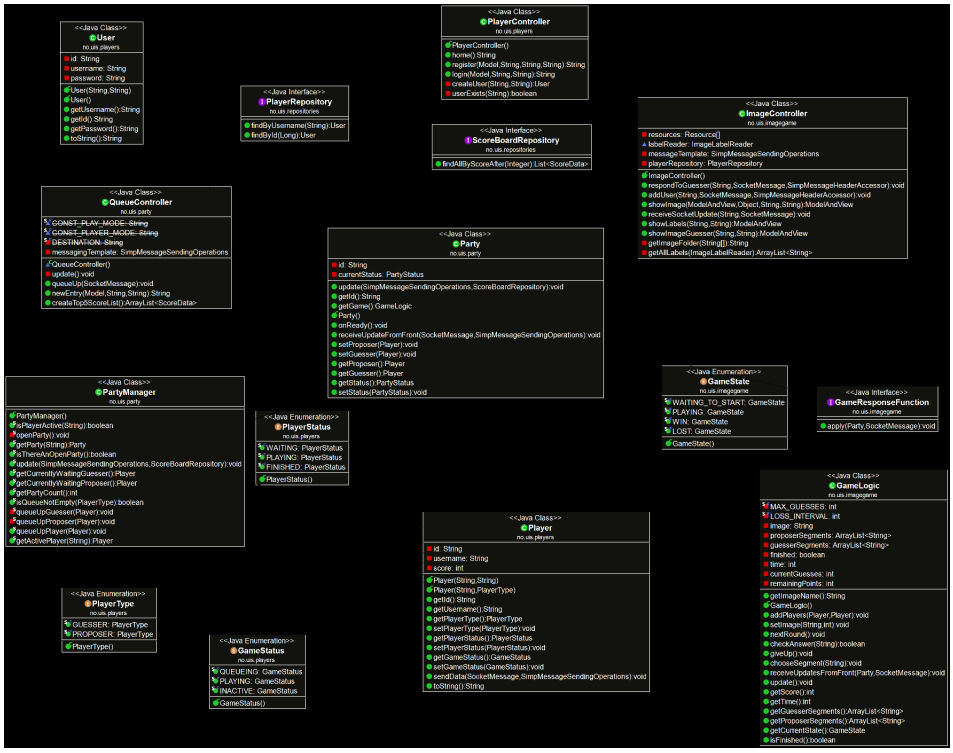


Figure 3.1: Class diagram for back-end architecture

The back-end architecture consists of classes that hold the internal logic of the application in an authoritative server model. All logic for the application occurs only on the server side (back-end) where the front acts like a receiver and sender of commands.

The coding procedure started with defining classes as the building blocks of the application; player, party, game logic, and party manager. All of which are classes that were built and tested through scanners and console inputs along with unit tests in order to achieve the core logic of the back end architecture. The classes act as entities that interact with each other in their respective context (e.g. players in a party playing a game handled by some game logic).

When the implementation of the communication technology (such as websockets, controllers and HTTP request handling) was completed, the refactoring of the key component classes begun to achieve compatibility with those implementations, rendering the game playable.

Please see figure A.8 in appendix for bigger class diagram.

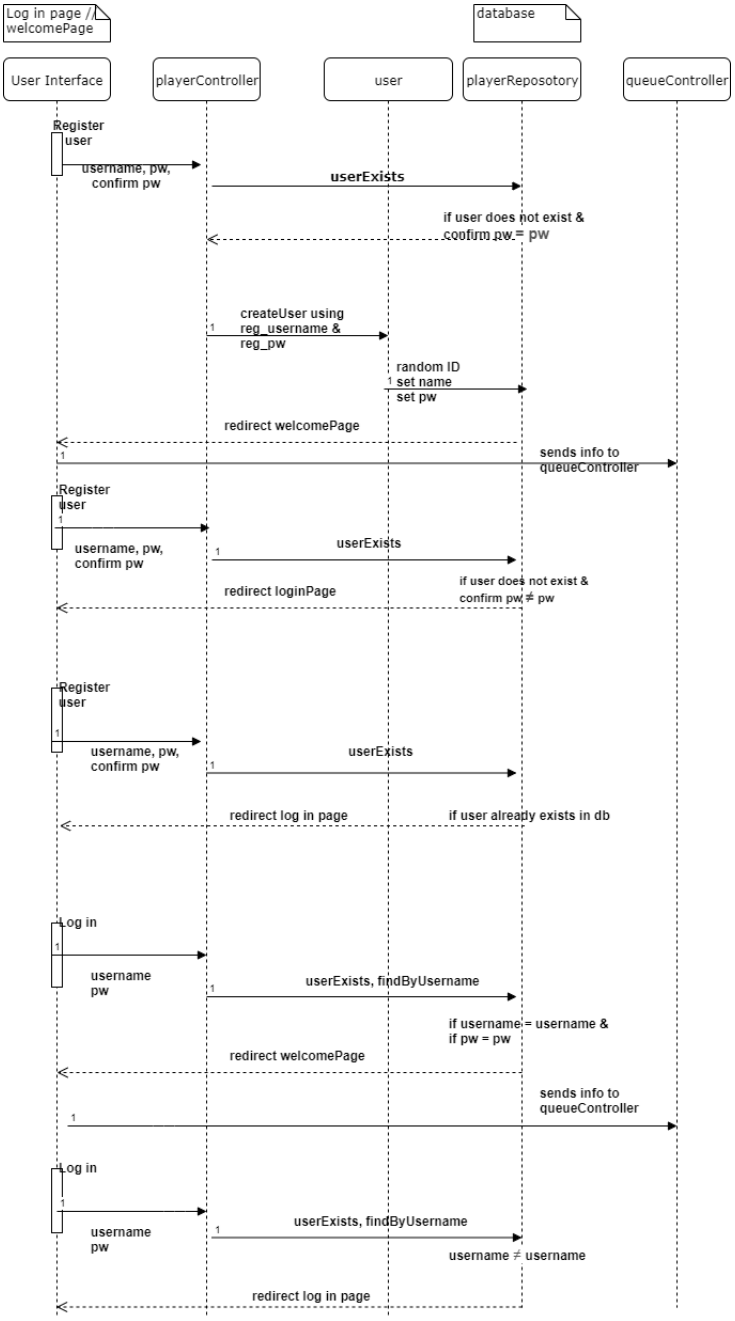
3.2. PLAYER CONTROLLER

A player controller was implemented to handle the creation of new users and log ins. The player controller reads the requests for registration of users from the registration fields in the login page and adds the user to the database if it doesn't already exist. If the request to create a new user passes the checks, the user is created and then redirected to the welcome page where the info is passed on to the queue controller.

If a user tries to register, but fails to either confirm his password or come up with a unique username. The player controller handles this issue by redirecting the user back to the login page with a message telling him what went wrong.

The welcome page also has a log-in field for already created users. When a user tries to use the log in field the player controller searches in the player repository to see if a user with the specific username exists in the database. If the username and the password are correct for the specific user, then the player controller redirects the user to the welcome page.

If the player controller fails to find a corresponding username to the log in field within the player repository, the user is redirected to the log in page. Below is a sequence diagram showing the player controllers interaction with new/current users, player repository for saving users and redirecting to the queue controller. See fig 3.2.



=

Figure 3.2: Sequence Diagram for the player controller

3.3. QUEUE CONTROLLER

The queue controller handles the action of the welcome page after the user has logged in. The controller receives the information from the front-end regarding the player role and its player mode. This information is forwarded to the party manager in form of a player object where the role "PROPOSER" or "GUESSER" are assigned. Their player statuses are set to waiting and the tick execution thread updates the queue every second to see if it can find a pair of available proposers and guessers. The party is then created and the proposer is sent to the proposer image selection view in image controller to select an image.
The guesser is set to waiting until the proposer has chosen an image and then the image controller initializes the game and sends a message to the guesser via the party class for information regarding the game and socket channel, which redirects the guesser to the guesser view.

Since each party has a reference to both of the players contained within it, (given each player object can send messages to the front) it has been given the ability to compose socket messages and send them to each player via references to message sending operation objects provided by the Spring websocket dependencies.

While simultaneously containing the game logic, party objects are the ones to set the players user permissions regarding their actions on the front-end via websocket messages. These messages contain information about the game logic handled on the back-end.

Below is a sequence diagram[2] (fig 3.3) explaining the internal flow of going from the queue to a new game involving the queue controller, party manager and party to the image controller.

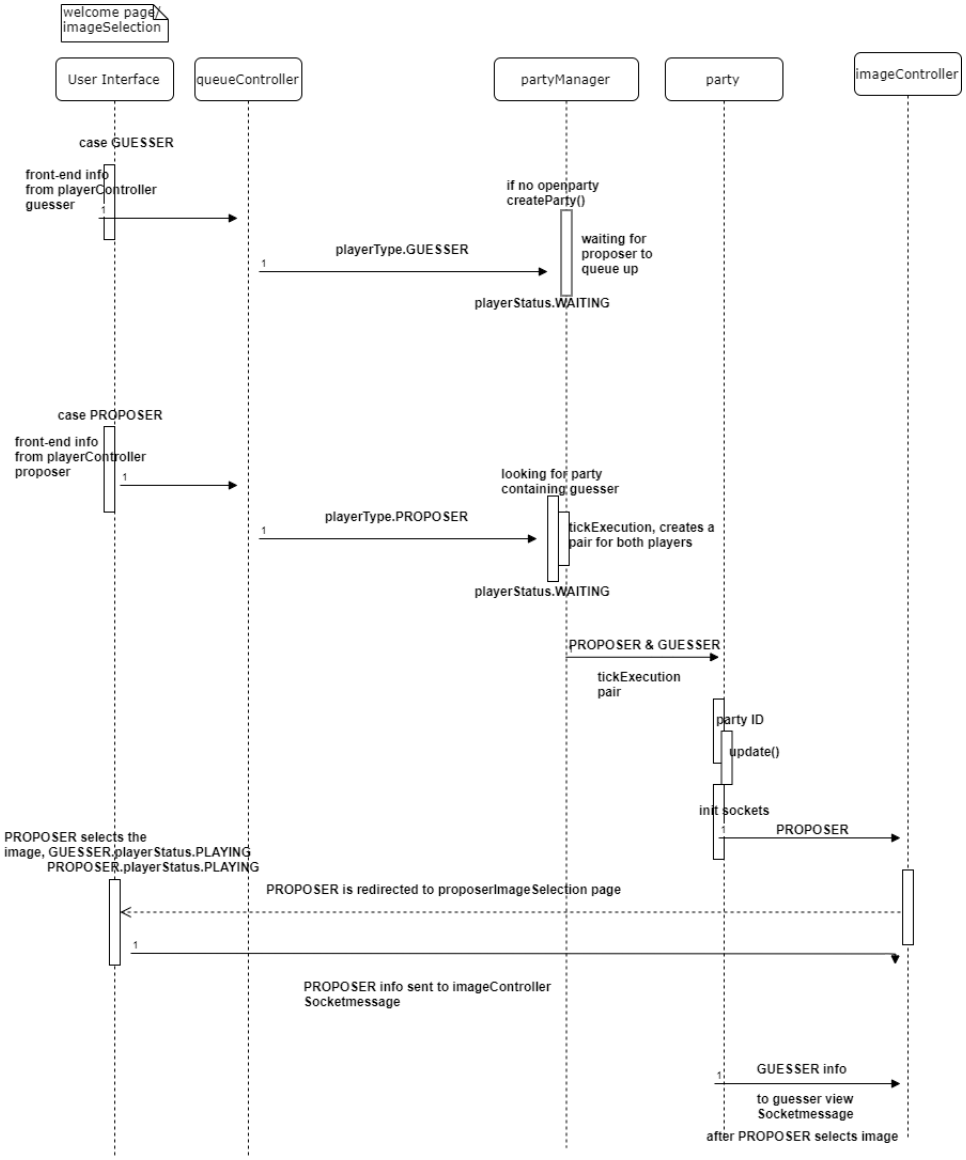


Figure 3.3: Sequence Diagram for the queueController

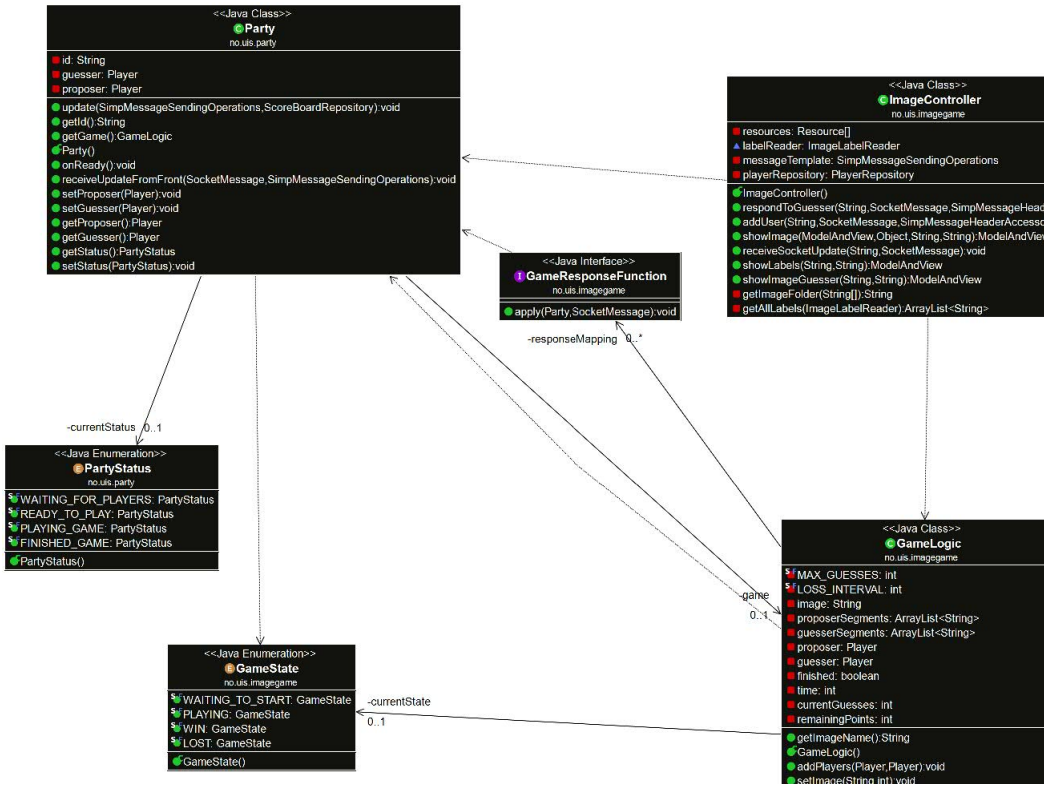


Figure 3.4: Class diagram for the central game functionality

The actual game in this project is mainly implemented over the following classes: party, game logic and image controller. The party object is created with an id that serves as the game identifier, and the game-logic is composed within the party class. The game logic processes the incoming messages from the party object and responds to the player object through the party with further instructions. The player object forwards the message to the front-end and sets the user permissions for the next action, such as who's turn it is and if they are allowed to make a guess or choose a new segment. See the figures above for the allowed game and player states.

The image controller handles the front-end view and receives user input from the front-end through websockets. Once a party joins a new game, the proposer is sent to the proposer image selection view, where it decides an image label for the specific instance of the game. Once a label has been selected, the image controller initialize the game by uploading the required resources for the players and sends the proposer to its proposer view. The controller sends a message to the guesser through the party object with initializing information for connecting to the same game and socket channel, in addition to time, score and selected label before redirecting the guesser to its guesser view.

The game logic processes the incoming messages and based on the mentioned game states, it will respond to the players through the party object which is then sent to the front-end where it sets the player permissions according to the game state. This means that requesting a new segment would be sent from the image controller to the party, processed by the game logic and it responds to the players through the party object which is then communicated to the front-end and restricts the guesser from making a new guess and allows the proposer to choose a new segment.

4. FRONTEND

Persons in Charge; Grégoire, Withya and Jari

The framework used for the front end was Thymeleaf [6] html templates. The communication from front end to the controllers used WebSockets to allow for dynamic functionality of the game.

4.1. SUMMARY : FRONT-END DESCRIPTION

The Front-End is composed by five views : loginPage, welcomePage, proposerImageSelection, proposer and guesser.

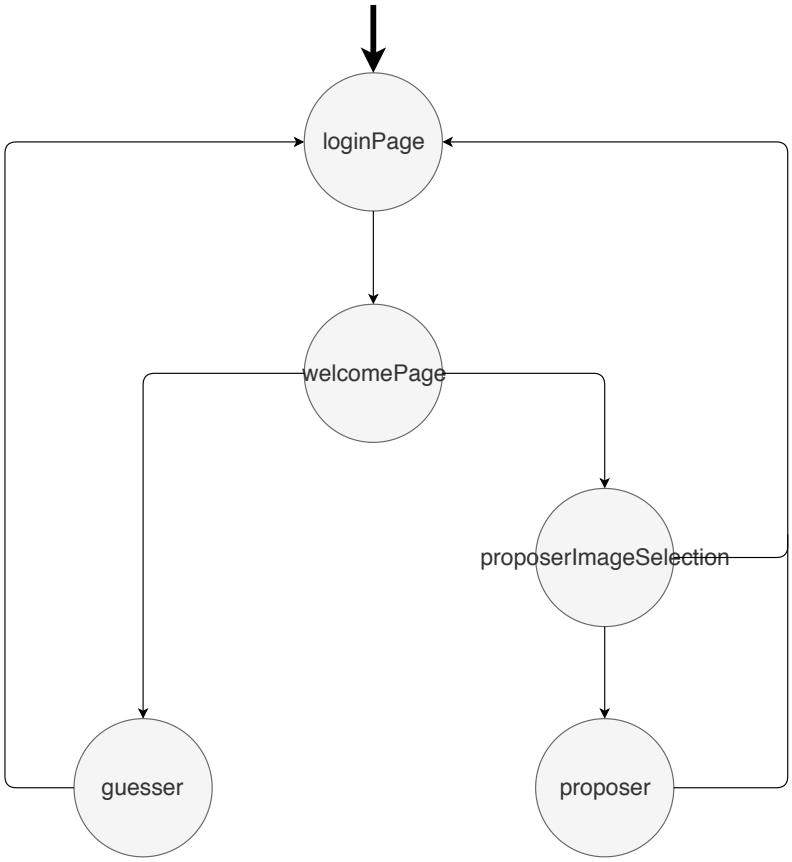


Figure 4.1: Links between the web pages

loginPage is the entry-point. It allows the user to create an account or to log-in to an existing one.

welcomePage is the view where the users can configure the party and then get access to the party waiting queue. It allows the user to set the game options and to see a leader-board displaying the best players in the game. The user can choose the player mode and the role he wants to play. Then he is allowed to lunch the party research. When the party is created, the user is renewed automatically to its game view according to its role. The user are displayed the leader-board while waiting for a party.

proposerImageSelection is the first view that a proposer gets access to in the game. It allows him to pick the image label that he wants for the specific game, which is then presented to the guesser. Then the proposer is redirected to the proposer view.

Proposer is the game view for the proposers. In this view, the proposer can pick which image segments that he wants to present for the guesser per turn. The proposer can see the guesser view, the guesser pseudo, the current score, the current time set, the game state and the highest score on this image. Moreover the user can leave the game and return to the welcomePage already logged or to the longinPage.

Guesser is the game view for the guessers. In this view, the guesser can send a guess or ask for an additional image segment. The guesser can see the proposer pseudo, the current score, the current time set, the game state and the highest score on this image. Moreover the user can leave the game and return to the welcomePage alreadyLogged or to the loginPage.

4.2. Front end design

In the following figures the view of the game from player perspective can be seen and the progress between views will follow the flow shown in figure 4.1 and explained in the subsection 4.1 above.

Figure 4.2: Login Page

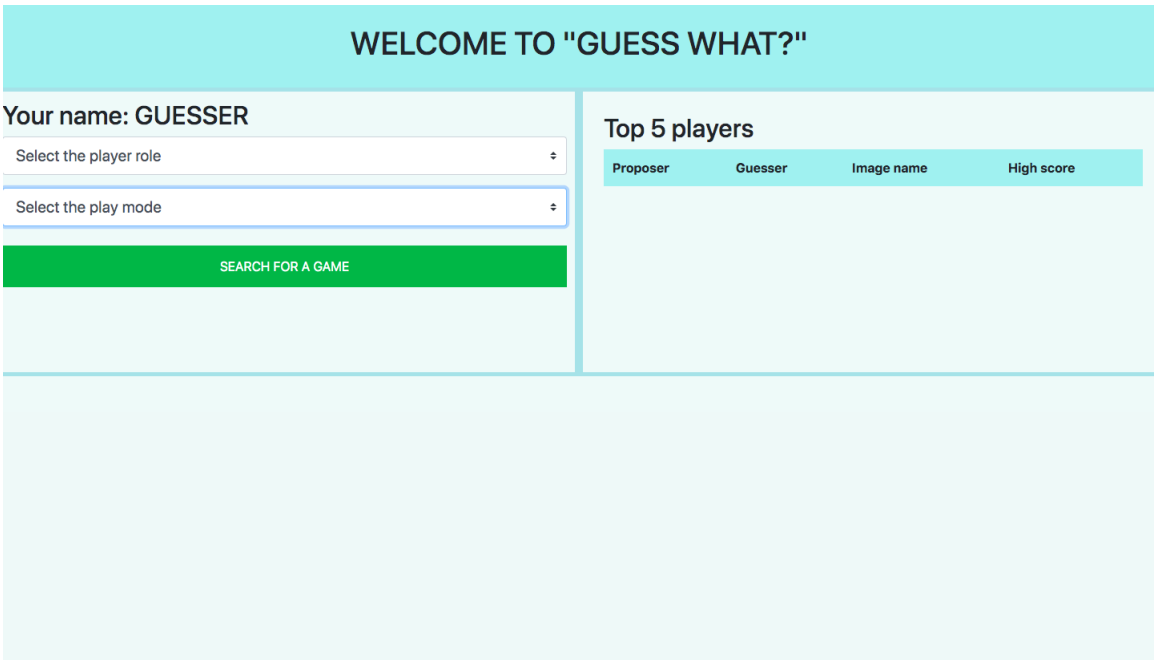


Figure 4.3: Welcome Page

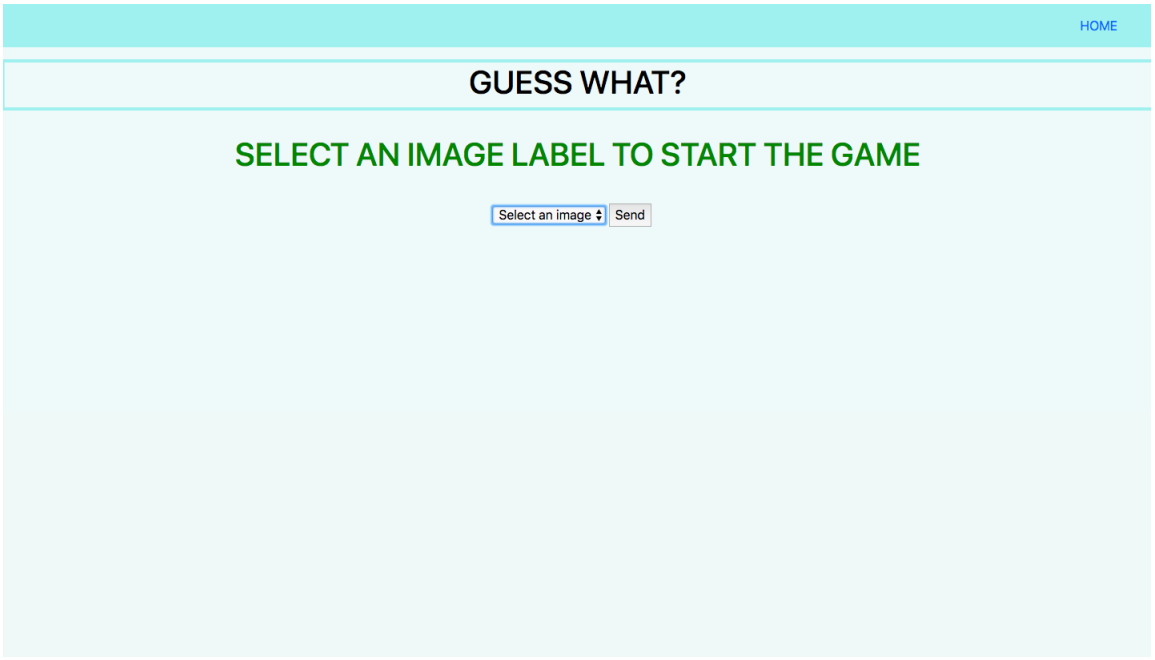


Figure 4.4: Proposer image selection

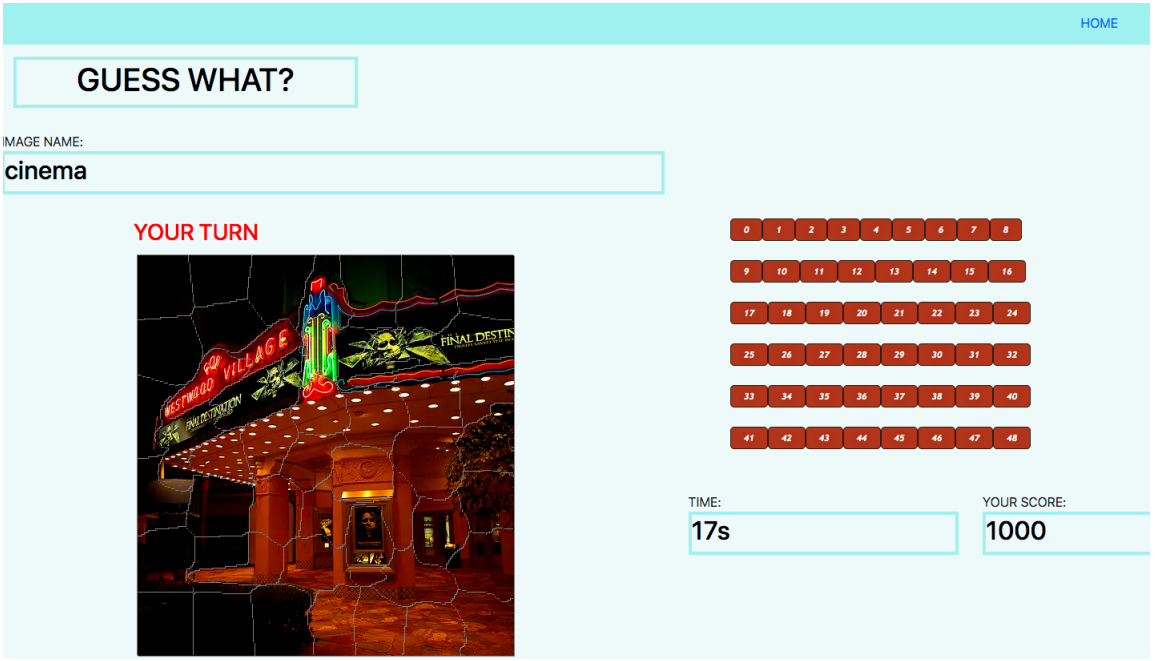


Figure 4.5: Proposer Page

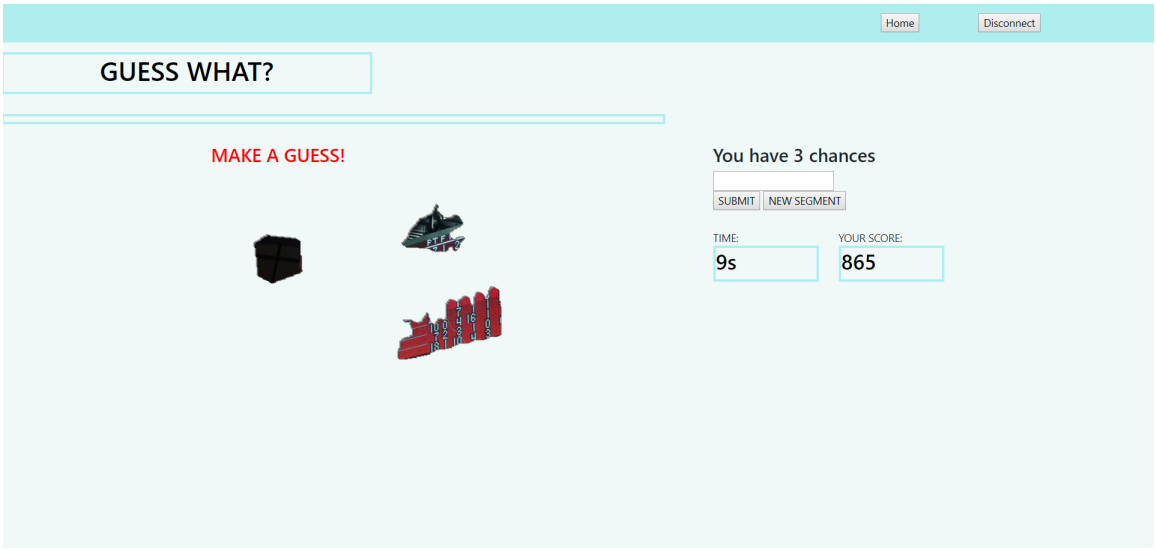


Figure 4.6: Guesser Page

4.3. HTML FILES

Each view has its own html[8] files where the view structure and the navigation access point can be found. The views are cut on a grid pattern where each html elements are placed. The variable elements are equipped with an unique id. The elements owns generic classes specified on a main css file. Thymeleaf were used to take back the images as well as the userId and partyId. This means that each player has an unique url.

4.4. JAVA SCRIPT EVENTS

Each view is linked to a specified JavaScript file which has the task to handle the different events (changing HTML statements based on their id) in the view to make it more responsive and pleasant for the user. For the game's view, a state manager carries the task to put the available HTML elements depending on the state of the game. (for example: the player can't make a guess if it's not their turn to play). JavaScript events are linked to the current state of the player. The views are updated depending of the state allowing (or not) the player to take action on the web page. The state can be changed depending of the user interaction on the webPage (for instance: the proposer will put the state at waiting when he will choose an image segment). It can also be changed by the back through the updates. Each variable element in the html file is set on the js file.

4.5. WEBSOCKET MANAGING

Communication between the player's view and the server is managed through websocket services. This service allows bidirectional messages between a server and a client thanks to an

open and persistent connection. It is one of the best ways to make a multiplayer application responsive.

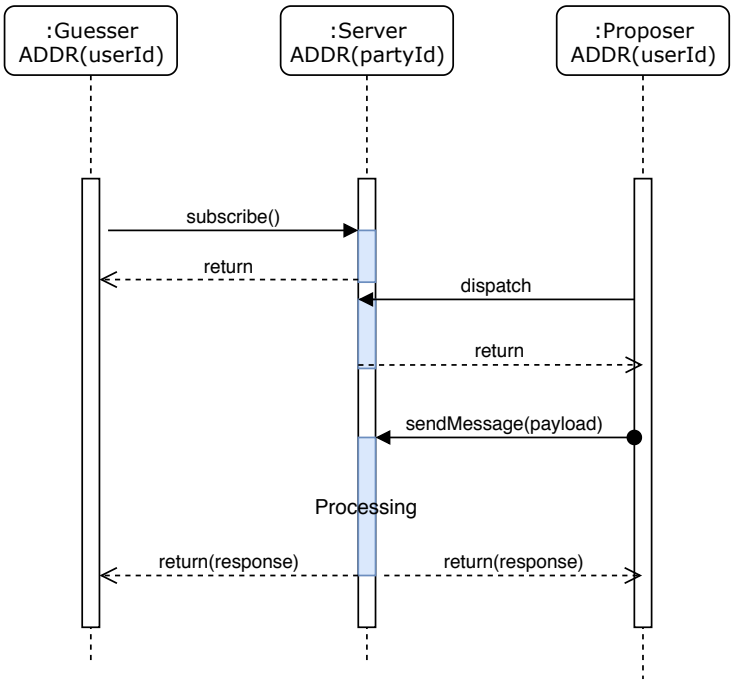


Figure 4.7: Websocket Principles

The websocket on the front is managed through the class called socket connector, which is abstractly the messaging protocol for the client. Firstly it creates the end-point on '/ws' address and stores the SockJs as the 'socket' attribute. The subscriptions attribute is a dictionary containing all the subscriptions that the client had applied. Stomp is the message listener, it carries the task to receive messages and to send them on the websocket. Stomp.connect is the connecting function which sends on the connections applied; they're stored on stompListenerQueue which is the queue of stomp connection applied in waiting of the servers. When a Stomp.connect has been approved by the server, the stomp is removed of the stompListenerQueue: The connection has been established. For the method: -addStompListener allows to create a new websocket connection with a server. -subscribe allows to subscribe the client to a specified address and to give the callback that the server should call. -unSubscribe allows to delete a subscription.

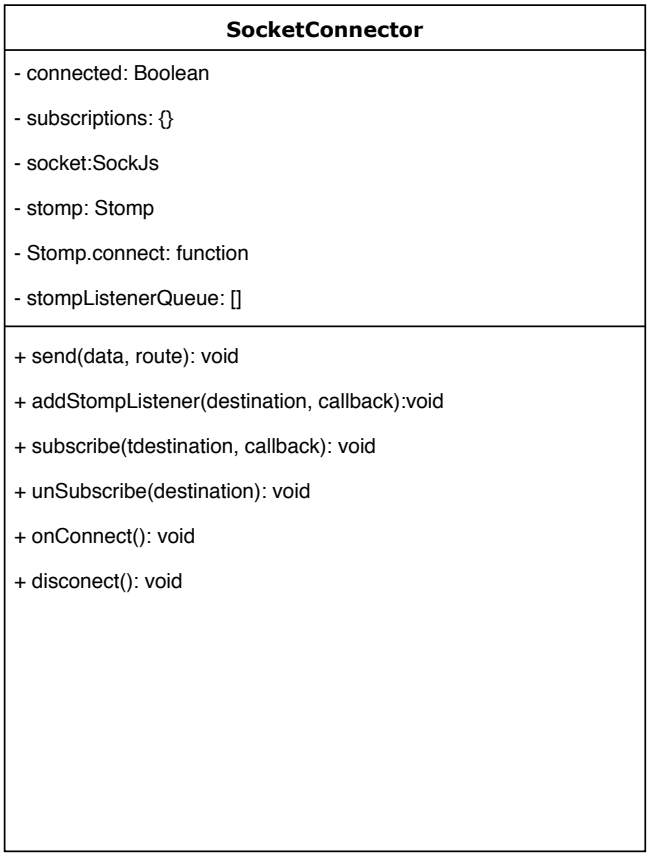


Figure 4.8: Connector-socket

Each player has an unique id, as well each party. When a player wants to send a message to the server, they send it to the adequate destination using the party id. This way the server can process the message and then update the view for each player in the party by sending a personal response to a destination depending on the user id.

The proposer and guesser have . For each file we will find a method called "subscribe" which subscribe the client to the server, Through this method the client give a callback function called update to the back. So the update method allows the back-end to update the players' views. The features updated are: -the state of the player (if he's playing, waiting or if the game is finished). -the current score. -the current timer. When the client wants to send information to the back, it uses the send method from Socket-connectors.Moreover their is a method called onConnect allowing the sending of a unique message when the websocket is created. It allows us to handle the page reloads and to send specific message: for instance, when the proposer has chosen the image, he arrives to the proposer view, this view connects to the back and send for first message that he's ready and the proposer can pass from the waiting queue to the proposer's view.

5. Organization

5.1. Agile Scrum Methodology

Scrum methodology[3] [7] was chosen to organize this project. Sprints were set at 1 week each and 4 sprints were conducted before submission. An alternative methodology, Kanban was discussed.The more strict structure of scrum was deemed the best alternative given the circumstance of limited experience and time for this project.

The general principles within the agile method were followed. Face-to-face communication was prioritized over communication via web. The group meetings were scheduled to be on campus. Either in the lab or in group rooms.

The group wanted to measure the progress based on creating working software in line with good practice for agile methodology. It was decided to focus on creating good design UML diagrams of the application. With the diagrams it would be clear for every team member of how the application would function. It would also be quicker to implement changes along the way, if better solutions were suggested. All of these diagrams are mentioned in the Design section [2] of this report and in the appendix A early design ideas can be found.

5.2. Pair programming

The decision of making use of pair programming was made because of the widespread of knowledge and programming experience in our group. There were no 5th semester students in this group, but some members had some background knowledge from personal experience. Therefore the most experienced members were combined with the ones with less experience. This would help towards creating the best possible outcome for this project.

The group was split into two teams, back end and front end team.

Backend team: Eirik, Markus and Alan
Frontend team: Withya, Grégoire and Jari
Project Owner: Grégoire
Scrum Master: Jari

Jari was selected as the project leader towards the end to be the communication link to the Professor. The scrum methodology sets up organisation without the need for the traditional leader to be in place so there were no extra tasks given to Jari as the project leader.

5.3. PROJECT BACKLOG

As the Project Owner, Grégoire was given the responsibility of the Project Backlog. A snapshot of the backlog can be seen in the figure 5.1 below. The intent of this backlog was to list up needed tasks or features that the project needed, as well as prioritize them for the groups sprint planning meetings or when someone finished their sprint task before the next spring meetings. A Google sheet was set up with the project backlog, time tracker(timesheets), burndown chart and the sprints to have as much as possible collected in one environment. These are demonstrated later in this section.

To Do List	**Project Backlog**	1,2,3,5,8,13,21,34,55,89				Product Owner: Gregoire		
ID	**Story / Tasks / Feature**	**Work Remaining**	**Priority[1-x]**	**Status**	**Skills requirement**	**Acceptance Criteria**	**Sprint #**	**Initial Estimation of**
1	What do we want to make?			Done		All assignement tasks understood and broken down into tasks	1	21
2	What do we need in front end?			Done			1,2	8
3	What do we need in back end?			Done		All requirements for the game to work mapped out	1,2	8
4	How do front and back communicate?			Done			1	13
5	UML diagram of the game			Done		UML diagram using a software package and all of the game mapped out	1	34
6	Set up teams			Done		All Members of group put into a team	1	3
7	Responsive front end design		low	Not Started				21
8	Assign Scrum Master and Product Owner			Done		Scrum master(Jari) and product owner(Gregoire) assigned	na	1
9	Set up documents and lists for scrum methodology			Done		Product backlog, time tracker, sprint backlogs, powerpoint presentation detailing work progress, code documentation	1	5
10	How do we approach unit design?			Done				5
11	How do we approach Coding?			Done				5
12	How do we approach requirement gathering?			Done				5
13	How do we approach Design process?			Done				5
14	As a user i want to log in and create a user to get access to the game.	2	6	In Progress				13
14.A	Have front object to log an account	1		In Progress				5
14.B	Implement back to access / create an account and change its state to LOGGED	5		Not started				5
15	As a proposer i want to show pieces of an image to my counterpart guesser.	8	4	Not Started		Images must transit without lags, bugs ect.. Test are expected		8
15.A	Front: Continue the interface to choose one part for the image	6		In Progress				6
15.B	Back: Send the image choose to the Class party	5		Not Started	Data manipulation (don t send the image but its address)			5
16	As a pair of user and guesser we want to have a leaderboard to see how good/bad we did compared to others.	5		Not Started				5
17	As a user i want to be able to log out.	3	7	Not Started				3
18	As a guesser i want to see the images that my counterpart proposes to me.	8	5	Not Started				8
18.A	Front: Concatenate the images available		-	Done				
18.B	Back: Take back the image available	3		Not Started	Data manipulation			3
19	As a proposer I want to give hints to the player	5	Low	Not Started				5
20	As a guesser/proposer I want to see the time since start	5	9	Not Started				5
21	As a guesser/proposer I want to see our current points	3	10	Not Started				3
22	As a player I want to be able to play agains a machine (choose random segments)	13	Low	Not Started				13
23	Check if project todo list can be part of the tasks for scrum			Done		decision made from Jari of how we sort the backlog	2	2
24	Create the design goal of each view (Front)			Done				2
25	Find out what we need to deliver for the project, and inform about group status			Done		Get a definite answer from Vinay	2	1
26	Create DataBase to store user data	4	11	Not Started	SQL skills tutorial : https: //mediasite.uis. no/Mediasite/Play/d92cb269 7bb5427e9dba3346214c693 d1d? catalog=27fff3c4469d43d7b 9e137275ca0e47821			4
				Done	Architecture / Enum Partern	Have created the class with the main method		
28	Create Test files (Back)	2	1	Not started	Achitecture / Gradle skills	Create the test class		2
29	Manage webSobcket	3	3	In Progress	https://spring. io/guides/gs/messaging- stomp-websocket/			3
30	A) Front : Create java script allowing the subscribe on the web socket	3			Spring understanding	Manage to update the front.		3
31	B) Implements WebSocketMessageBrokerConfigurer	3			Spring understanding			3
32	add continous integration test to github	5		Not Started	tutorial (end of the video) : https://mediasite.uis. no/Mediasite/Play/d92cb269 7bb5427e9dba3346214c693 d1d? catalog=27fff3c4469d43d7b 9e137275ca0e47821			5
33	Sequence diagram of the game from the users view(high level)	3		In progress		All objectives and functionality of game covered		3
34	Get the backend game logic up and running including unit tests	8		Not Started		Tests are written, the game is runnable		8
35	JavaDoc Latex integration			Done		understanding on how you can use javadoc in latex is needed and then decision on how to use it for the report must be made	3	3
36	Create explaining class diagram for thread management	1		In progress				1
37	Improve the guesser/proposer front end pages	5		In progress			3	8
38								
39								
40								
41								

Figure 5.1: Project Backlog 2nd November 2019

5.4. SPRINTS

Scrum Methodology was the groups work framework. It was planned to have 4 sprints between the start of the project until delivery. A week was set per sprint which is not very long with the limited time available to work on the project every week. 4 sprints ended up be-

ing the selection so there could be closer track on progress.The project group was divided in two separate teams called "Team Kjølv Egeland" (back-end) and "Team Arne Rettedal" (front-end). Various tasks were selected by each team/developer per sprint.

5.4.1. Sprint 1, 09.10.2019 - 15.10.2019

In the first sprint, the back-end team and the front-end team picked the tasks of figuring out what was required in the two different branches for them to be able to work as two separate stand alone projects to maintain the agile methodology. Techniques and methods onto how the front-end and the back-end in the future would communicate as well as establishing an idea for exactly what product the group as a unit wanted to produce.

The back-end team were assigned the task of figuring out exactly what the group needed to be able to solve the game problem. The front end team was also assigned the task of making the UML-diagrams mentioned in (2) DESIGN part, needed for the project.

Sprint1 Backlog	**Start Date**	**9.10.2019**	**End date**	**15.10.2019**
Team Kjølv Egeland				
ID	**Story**	**Work Remaining**	**Priority[1-x]**	**Who**
3	What do we need in back end?	8	High	Markus, Alan, Eirik
4	How do front and back communicate?	13	High	Markus, Alan, Eirik
1	What do we want to make?	21	High	Markus, Eirik, Alan
Team Arne Rettedal				
ID	**Story**	**Work Remaining**	**Priority[1-x]**	**Who**
2	What do we need in front end?	8	High	Gregoire, Withya
4	How do front and back communicate?	13	High	
5	UML diagram of the game	34	High	Gregoire
9	Set up documents and lists for scrum methodology	5	High	Jari
1	What do we want to make?	21	High	Jari, Gregoire, Withya
6	Set up teams	3	High	Jari, Eirik, Withya, Gregoire, Markus

Figure 5.2: Sprint 1 Backlog

5.4.2. SPRINT 2, 16.10.2019 - 23.10.2019

In the second sprint, we figured out that we still had some work left on establishing all the parts needed for the two branches to work as stand alone projects. We also came to the conclusion that we were somewhat unsure of which elements to prioritize in order to achieve the best grade as possible. We decided to send Vinay a message for some clarification. The response lead us to prioritize a functioning game written using good coding practice as well as trying to follow the syllabus of agile methodology.

Sprint2 Backlog	**Start Date**	**16.10.2019**	**End date**	**23.10.2019**
Team Kjølv Egeland				
ID	**Story**	**Work Remaining**	**Priority[1-x]**	**Who**
24	Player objects, guesser and proposer	21	High	Eirik, Markus, Alan
3	What do we need in back end?	8	High	Eirik, Markus, Alan
Team Arne Rettedal				
ID	**Story**	**Work Remaining**	**Priority[1-x]**	**Who**
23	Check if project todo list can be part of the tasks for scrum	2	medium	Jari
25	Find out what we need to deliver for the project, and inform about group status	1	medium	Jari
2	What do we need in front end?	8	high	Withya, Gregoire, Jari

Figure 5.3: Sprint 2 Backlog

5.4.3. SPRINT 3, 23.10.2019 - 30.10.2019

For the third sprint, the back-end team were assigned the get the game logic in place as well as tests to back up the code. The front-end was assigned tasks such as improving the proposer and the guesser pages, as well as making additional diagrams for easier understanding of the application. It was also concluded that the communication between the proposer page and the guesser page were to take place by implementing usage of websockets. Therefore we decided that we from now on wanted the front-end and the back-end to cooperate during our planned work sessions. This decision was made as an effort to try to enhance our groups efficiency and understanding.

Sprint3 Backlog	**Start Date**	**23.10.2019**	**End date**	**30.10.2019**
Team Kjølv Egeland				
ID	**Story**	**Work Remaining**	**Priority[1-x]**	**Who**
30	A) Front : Create java script allowing the subscribe on the web socket	3		Alan
31	B) Implements WebSocketMessageBrokerConfigurer	3		Alan
34	Get the backend game logic up and running including unit tests	8		Markus, Eirik
Team Arne Rettedal				
ID	**Story**	**Work Remaining**	**Priority[1-x]**	**Who**
30	A) Front : Create java script allowing the subscribe on the web socket	3		Gregoire
32	add continous integration test to github	5		Jari
33	Sequence diagram of the game from the users view (high level)	3		Jari
35	JavaDoc Latex integration	3		Jari
37	Improve the guesser/proposer front end pages	8		Withya

Figure 5.4: Sprint 3 Backlog

5.4.4. SPRINT 4, 30.10.2019 - 03.11.2019

With this final sprint meeting it was established that we still had a lot of work remaining on our report. It was urgent that we started on writing is as soon as possible. Our application did also contain some bugs that we had to manage. We were having issues with the WebSockets implementation which led to that the guesser page only received the first image segment from the proposer page. All of these tasks were assigned both teams.

Sprint4 Backlog	**Start Date**	**30.10.2019**	**End date**	**03.10.2019**
Team Kjølv Egeland				
ID	**Story**	**Work Remaining**	**Priority[1-x]**	**Who**
38	Working on the report in order to get done before deadline	34	Urgent	Eirik, Alan, Markus
39	Fixing the various bugs in the application	34	Urgent	Eirik, Alan, Markus
40	Managing the websockets correctly	34	Urgent	Eirik, Alan, Markus
Team Arne Rettedal				
ID	**Story**	**Work Remaining**	**Priority[1-x]**	**Who**
38	Working on the report in order to get done before deadline	34	Urgent	Gregoire, Withya, Jari
39	Fixing the various bugs in the application	34	Urgent	Gregoire, Withya, Jari
40	Managing the websockets correctly	34	Urgent	Gregoire, Withya, Jari

Figure 5.5: Sprint 4 Backlog

5.5. BURNDOWN CHART

A Burndown chart was set up according to Scrum practice. The intent of this chart is to be able to predict progress of software projects. Since the time period for this project was relatively short and therefore the benefit of this chart was limited or close to 0. The figure 5.6 below show that the group identified more new tasks needed than it was able to do. So the remaining effort did not noticeable decrease until the last sprint.

Sprint	1	2	3	4
Total Effort	134	175	269	371
Remaining Effort	134	120	130	60
Burndown Veloci	0	14	-10	70

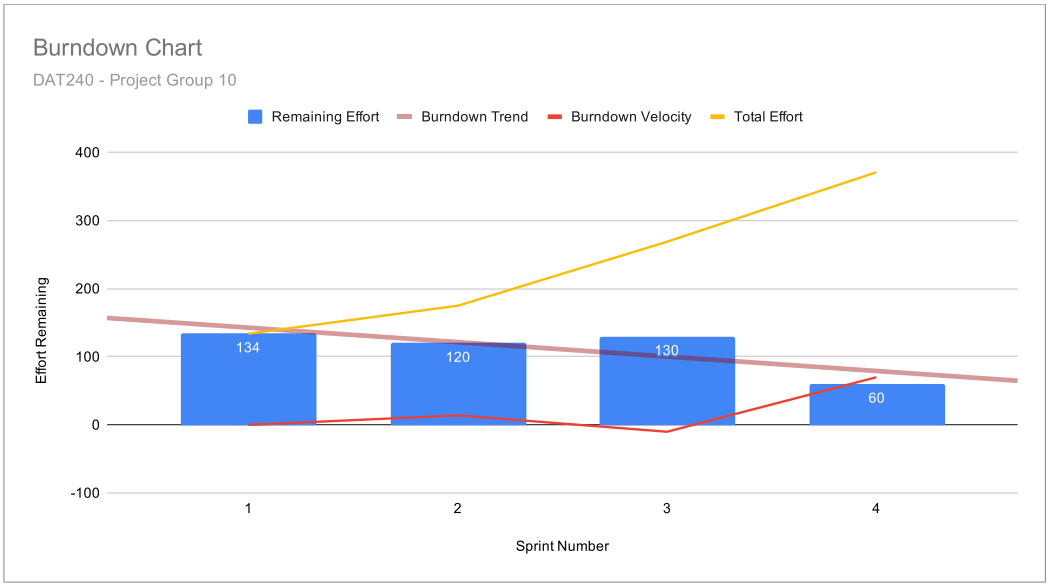


Figure 5.6: BurnDown Chart

5.6. Collaboration applications

5.6.1. Slack

Slack was selected as a communication hub for sending links and keeping the team members updated. An example from the Slack group can be seen below in the Figure 5.7 below.

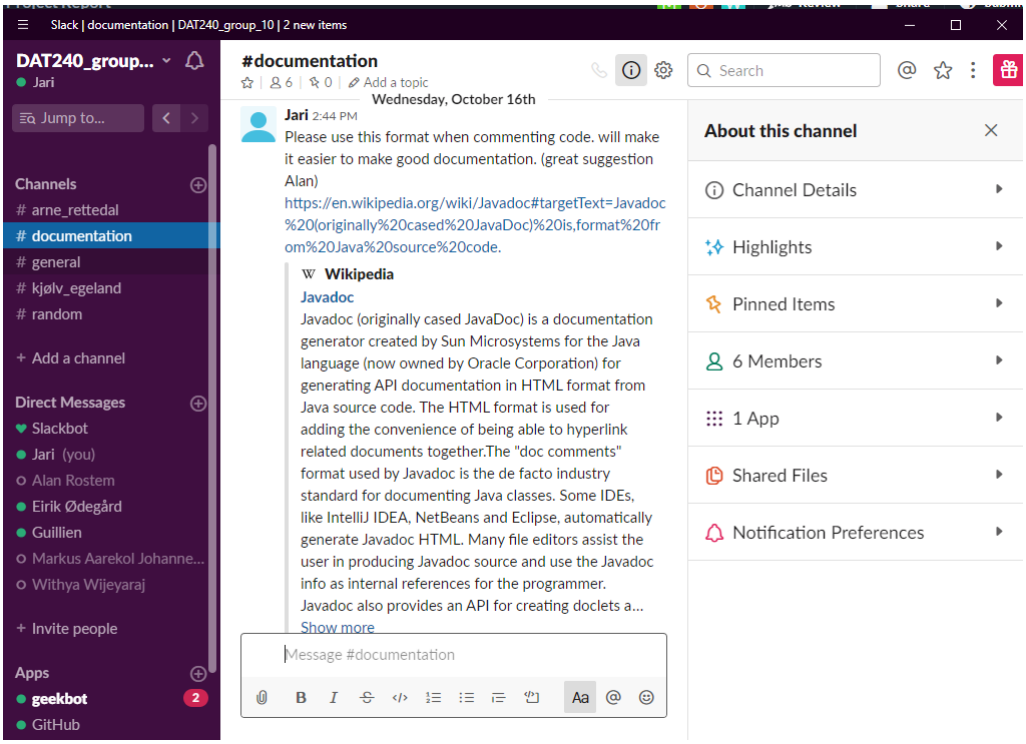


Figure 5.7: Slack for Group 10

There was set up a daily standup bot(Geekbot) that allowed for remotely to do scrum daily standups. See figure 5.8 below for an example of one of the standups.

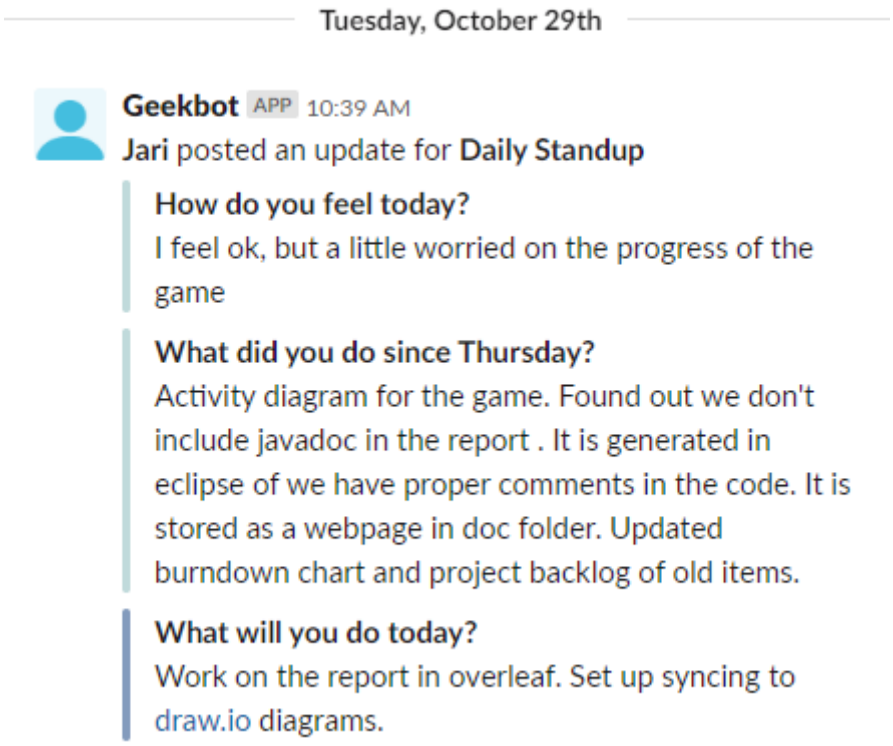


Figure 5.8: Daily Standup Example

5.6.2. Google Drive

Google Drive was used to have some shared folders for different files, the google documents and UML diagrams was hosted from there.

5.6.3. Draw.io

Draw.io[5] was used to produce some of the UML diagrams. It is an open source application with functionality to make the diagrams needed.

5.6.4. GitHub

Github [1] was chosen as the version control repository for this project. This allowed for collaboration and was the repository the group members had experience from before. Branches for backend and frontend was continuously maintained with regular updates to the master once there was something of significance made or useful for the other end to have available for coding. Using the Project functionality was explored, but a conclusion was made that the workflow there seemed more in line of Kanban Agile methodology while the group had selected Scrum methodology. It was decided to keep to the sprints and product backlog in the google sheet instead of the Kanban style to do board available in Github.

6. Summary and Contributions

6.1. Summary of the project

A working game according to the specifications was made within the time limit. The game exceeds features needed according to what was the basic assignment. The use of Websockets has allowed for a dynamic game. User creation and authentication was implemented and the applications is able to host multiple parallel games.
We used scrum as our agile methodology for this project. Looking back at our sprints and the work that we got done on our project. We may conclude that the scrum approach went somewhat decently. For each sprint it was very clear for each member what tasks that were to be done until the next sprint. We did also manage to maintain the agile mindset throughout the entire project, thanks to scrum.

We did however have some struggles with the sprints being only 1 week apart. For each sprint there was a lot of work that had to be done. Therefore we were not able to restrict the workload between every single sprint. Which conflicts with the sprint definition, which says that sprints are considered to be complete when the time period expires.

Due to some circumstances that had nothing to do with us directly, our team got restricted to 6 members already by the first sprint. All 6 of us were also 3.semester students and had no background knowledge from the Database, or Web programming courses. Despite this, we are left with the impression that we still managed to provide the best product with the resources we had available.

6.2. Contributions

Each chapter in this report starts with "Person in Charge: ...". This is a good guide of where personnel contributed. The JavaDOC and commits in the GitHub repository [1] show who coded the specific things even though a lot of co-located collaboration is behind the individual code and commits. Researching the functionality of spring boot [4] and other framework and tools is not recorded in this report as contributions, but acknowledged as an important contribution. The table 6.1 below lists the main products and elements of the project.

Table 6.1: Contributions to the DAT240 Project- Group 10

What	Who
Front end HTML(Welcome Page, Guesser and Proposer view	Withya
Front end CSS	Withya
Front end Javascript	Grégoire
Front end HTML(login Page)	Jari
Front end WebSockets	Grégoire
Back end WebSockets	Alan
Back end Game Logic	Eirik and Markus
Project Owner - Project backlog maintenance	Grégoire
Diagrams	Grégoire, Markus and Eirik
Scrum Master - Srum/Sprint documents and meetings	Jari
Report	All

A. Appendix - Initial Design Ideas

[H] *Person in Charge; all*

A.1. Block Diagram

This block diagram depicts our mid project thoughts for the game design. Users are added to the game controller as a party object and communicates through the web socket implemented in the player class. Once a message is sent between the players, the web socket sends an update to the controller which refreshes the view by the tick execution thread.

The web sockets are implemented in layers as they have to communicate with different objects throughout the web application. The player class has a send data function which enables the players to communicate with messages, which is continuously updated in the party class. The party class passes on the message to the controller which is then updated by the tick execution thread.

The web socket messages keep track of its game and player states. A quick walk through would be for the guesser to receive a message that its their turn, the controller receives an update giving the guesser a new segment and allowing the guesser to write a guess. The game logic resides within the controller so once it receives the message, it changes the player permissions and updates view for the current round.

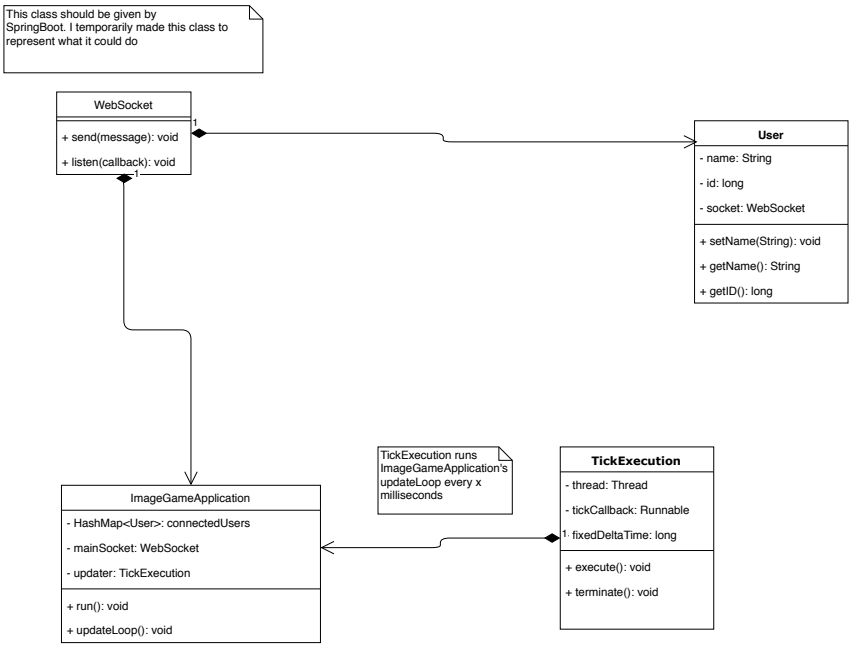


Figure A.1: Core Diagram Game

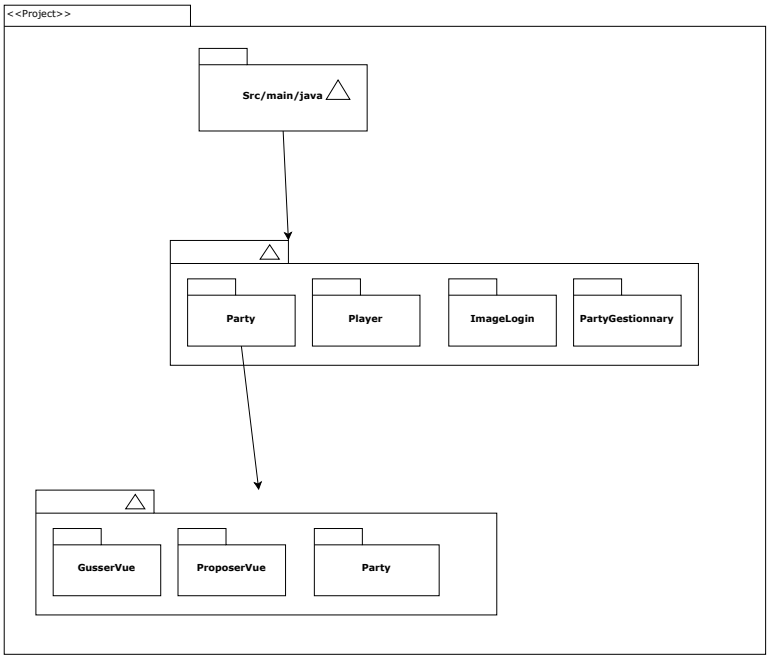


Figure A.2: Initial Files Architecture of our project

We also wanted to structure our classes in sensible packages to make the code more readable. i.g suggestions were made to separate logic for implementing players from the guesser/proposer views.

A.3. Initial Class Diagram

To begin with, we thought about a structure where the players (and the data linked to them) will be instanced on the environment. Firstly, PartyGestionnary is the core of the program, it has two important tasks: To queue the players and make them find a party and to create the party.

Player is the class working as an instanced database (on the waiting to create a real database). PartyController is the welcome view controller which should login the players and put them on the PartyGestionnary queue. Finally it should return the player on the play view. Party is the class containing all of the party information (like the image label, the score, the rount ect..). It's composed of both player controller allowing party to update the players view. It's composed of the two players allowing the party to update the players records.

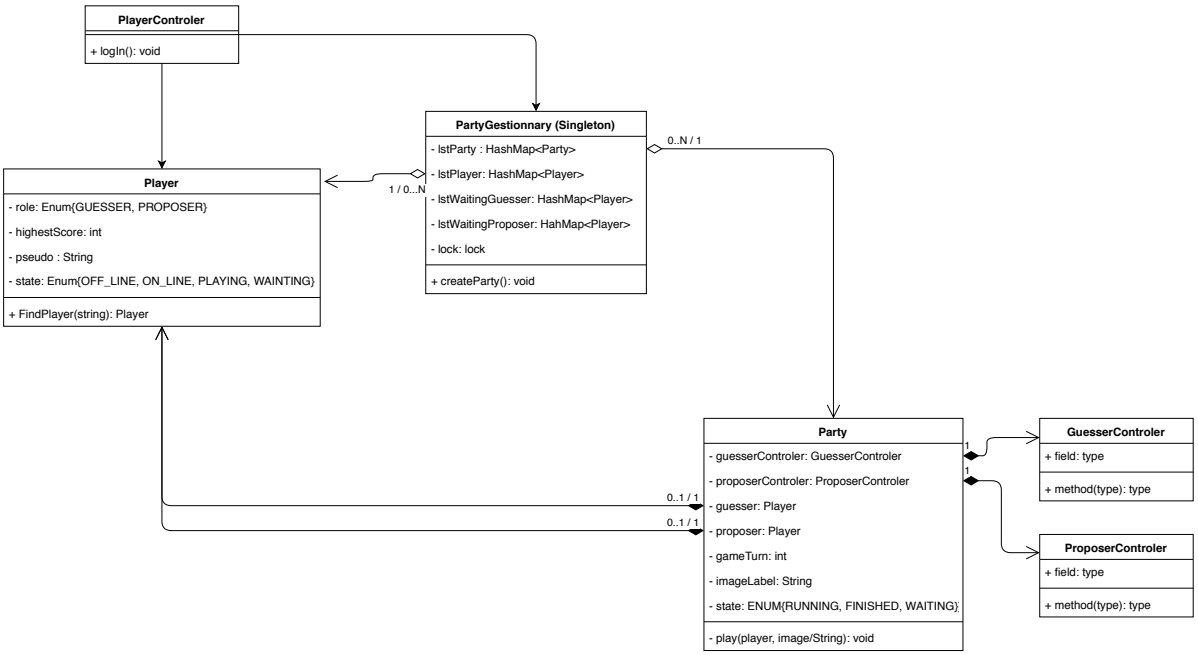


Figure A.3: Class Diagram Game

A.4. Early Sequence Diagram

This was an early plan on the sequence diagram.

The object instantiation order on function of the time is described on the figure A.4.
First, an instance of PlayerControler is created when a user is connecting to the welcome page. He logs-in which creates/takes back the instance of Player according to the log parameters. Then playerController asks PartyGestionnary to create add this object player to the waiting queue and to create a party when he match. PartyGestionnary create the party, return the url of the appropriate role controller (GuesserControler or ProposerControler) to playerController. Then the user access to the new url and play the party. When the party is finished, the instance of Party updates Player and the user is renewed to the view instanced of PlayerController. We can note that the instance of PartyGestionnary has been existing since the launch of the application. We can notice too that the instance of Player is never erased while the application is running. When we stop the application, we loose all information concerning the players. That's why it's an architecture basement. We should reconsider it with the project progress.

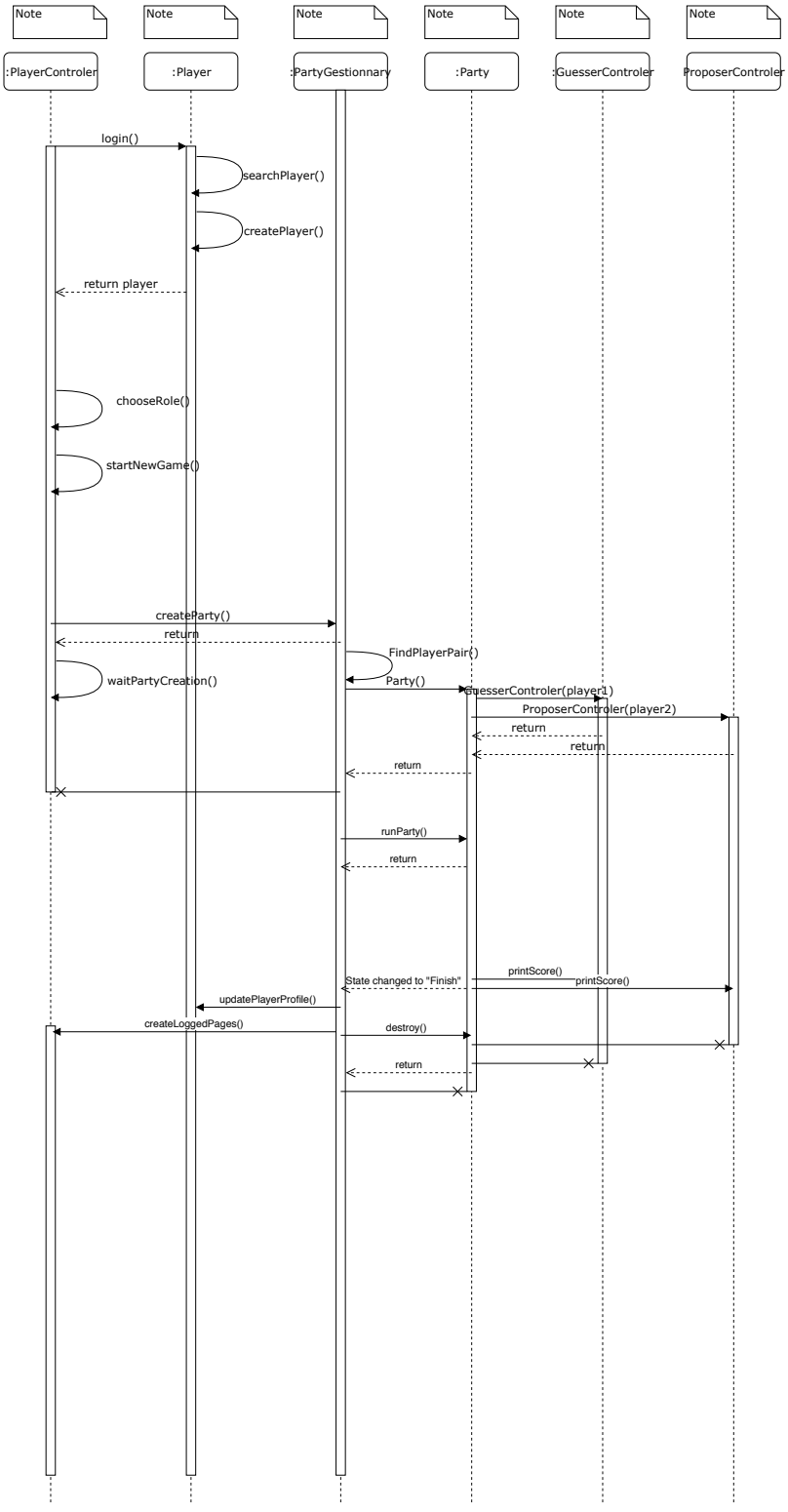


Figure A.4: Sequence Diagram

A.5. Initial Front Design

Before creating the HTML files, we decided to make some sketches of how we wanted the game to appear for the players. This made it easier for us to understand what needed to be implemented at the front end. To start with, our idea was three pages; welcomePage, proposer and guesser. The welcome page for login and game settings, the proposer page for choosing images and segments, and the guesser page for guessing labels and requesting segments. Getting these done at such an early stage made it easier for us to come up with suggestions onto how we could improve each view. This lead to an enhancement towards our agile approach for the project.

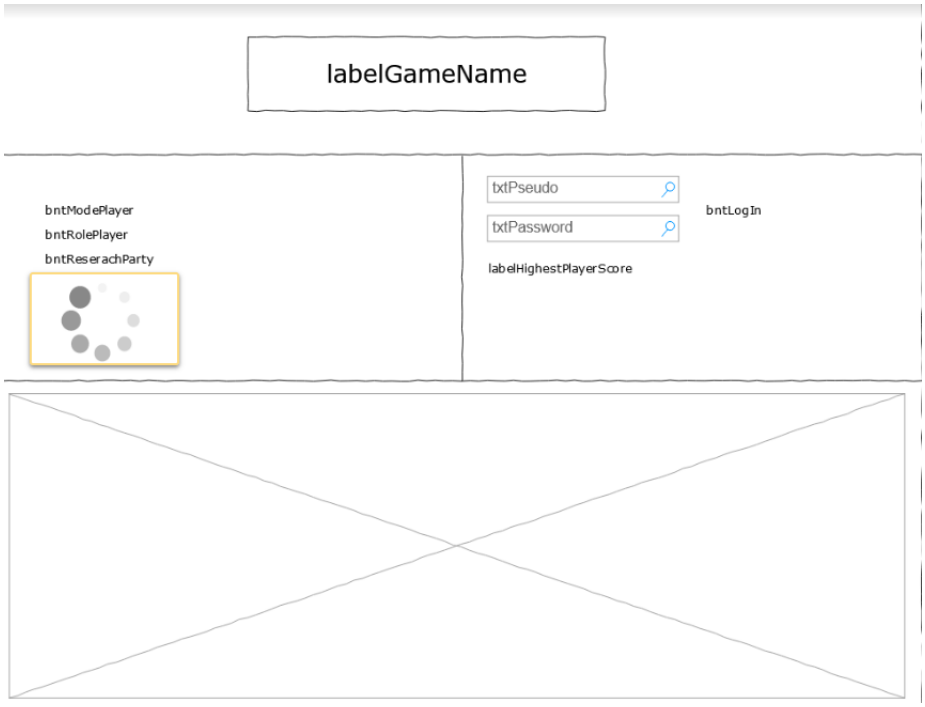


Figure A.5: Welcome Page View, example made in sprint 1

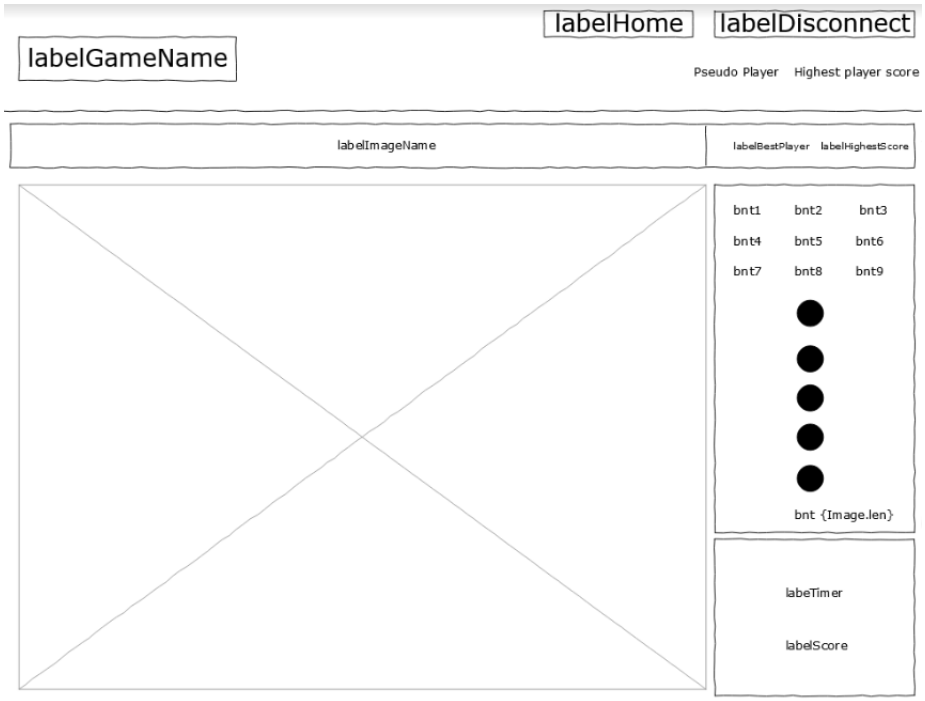


Figure A.6: Proposer Page View, example made in sprint 1

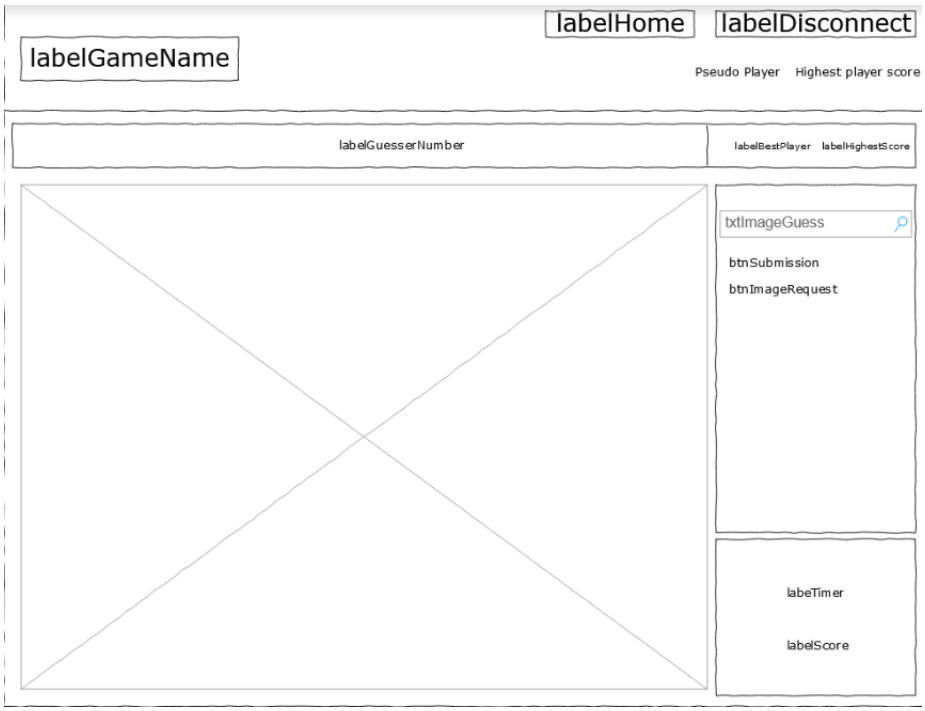


Figure A.7: Guesser Page View, example made in sprint 1

As we started to develop the game we decided to separate the login from the game settings. We therefore created the loginPage, as we wanted the user to be logged in to access game options. We also added another proposerImageSelection page where the proposer would have time to select an image for the game, before being directed to the proposer page where the game starts. In total, 5 templates were used for the game.

A.5.1. Final Class Diagram

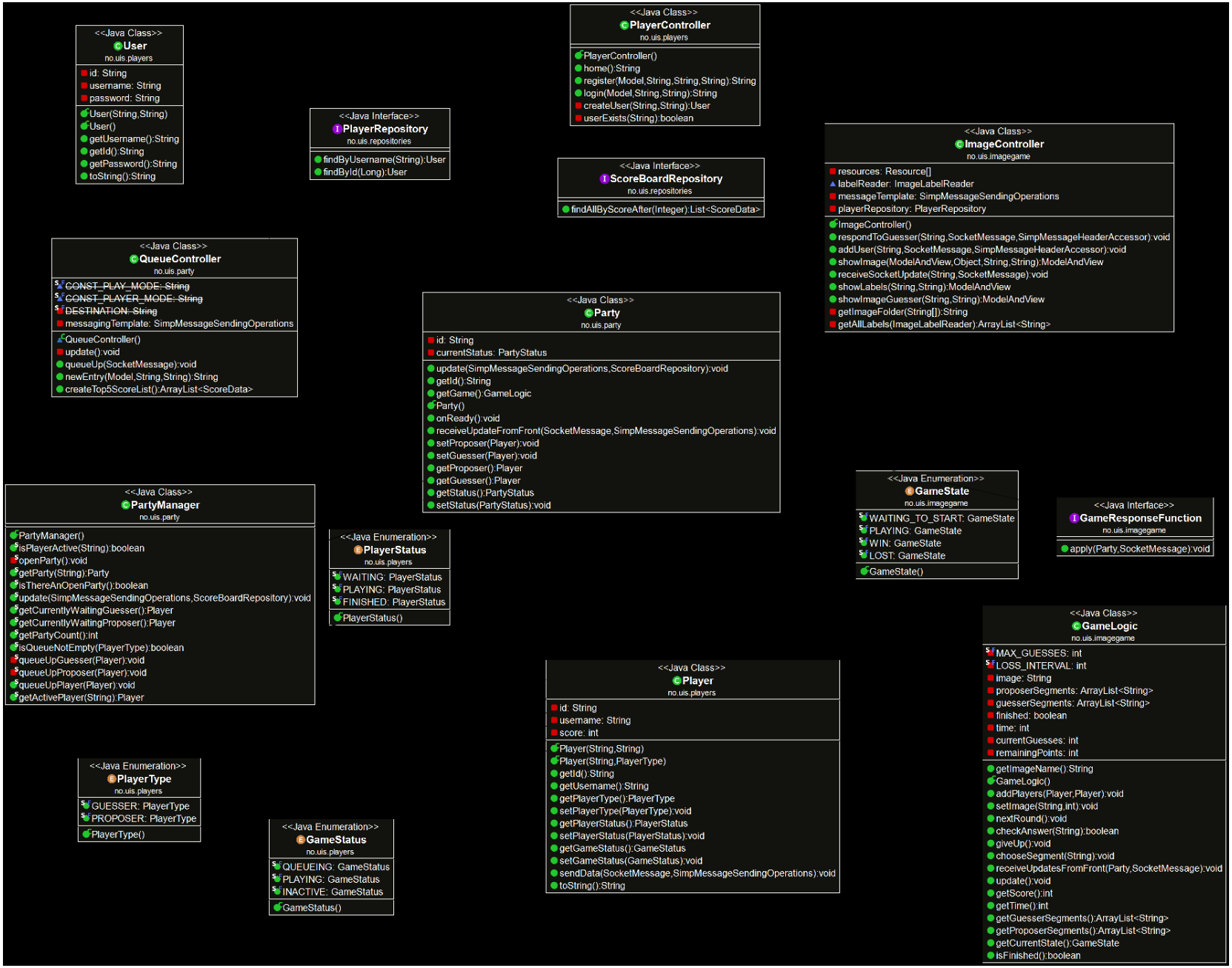


Figure A.8: Class diagram of back-end architecture

]

REFERENCES

[1] P. G. 10. Dat240 project repository. https://github.com/eirikod/DAT240-project, (2019).

[2] GeeksForGeeks. Sequence diagram. https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/, (2019).

[3] Linchpinseo. A beginners guide to the agile methods and scrum. https://linchpinseo.com/the-agile-method/, (03.11.2019).

[4] OpenSource. Java framework. https://spring.io/projects/spring-boot, (03.11.2019).

[5] OpenSource. Diagramming application. https://www.draw.io/, (2019).

[6] OpenSource. Thymeleaf - java template engine. https://www.thymeleaf.org/, (2019).

[7] Visma. Kort introduksjon til scrum. https://www.visma.no/blogg/en-kort-introduksjon-til-scrum/, (2019).

[8] w3schools.com. Html tutorial. https://www.w3schools.com/html/default.asp, (2019).