BATTLE BEASTS

GROUP IP1 | PSIT3 IT18a_ZH

Cyril Wanner
Marc Berchtold
Ilbien Paul
Marcel Brennwald
David Tschan

TABLE OF CONTENTS

USE CASES

USE CASE 1: REGISTER ACCOUNT

Main success scenario:
Not registered users can register themselves using a button that is clearly visible in the UI. The user has to fill in a unique username, a unique email address, a password with at least 8 characters and the password confirmation. The inputs are checked directly for their validity. The form can be sent by a button labeled “register” which is visible in the UI. If the registration is successful the user is redirected to the homepage.

Alternate scenario:
If the username or email is already used, a warning is displayed and the registration process is stopped. The warning displays the cause for the error. For example: “username already in use”, “email already in use”, “password have to be at least 8 characters long”.

USE CASE 2: DO LOGIN

A player who wants to play a game or manage his account can login using the form available in the website header. After entering his username and password and submitting the form, the credentials are checked against the database. In case of a successful login, the login state will be stored locally in the browser and the player gets a message saying that the login was successful.

USE CASE 3: DO LOGOUT

A player who wants to log out and leave the game can use the logout button which is visible in the UI. When the button is pressed a message will be displayed that the logout was successful. Logging out helps to protect the players account.

USE CASE 4: CREATE DECK

Scope: Battle Beasts - Browser Game
Level: User-goal
Primary Actor: Player
Stakeholders and Interests:

- Player: Wants an easy and fast way to create decks so he can choose which cards he wants to use in his next game.
- Company: Wants players to be able to create decks so they have more control over the game and can choose the cards they like most and keep playing the game with different strategies.

Preconditions:
Player is logged in and authenticated.
Postconditions:
New deck is saved with all selected cards and can be chosen for the next game.
Main success scenario:

1. Player navigates to the deck management page.
2. Player starts creating a new deck.
3. Player sees all his unlocked cards and the number of cards left he can put in his new deck.
4. Player selects a card which he wants to store in his new deck.
5. The card moves into his deck and the amount of cards which can be put in the deck gets updated.

Player repeats step 4-5 until his deck is full.

6. Player chooses a name for the deck.
7. Player saves the deck.
8. System validates the deck name and cards.
9. System saves the new deck with all cards in the database.
10. Player gets redirected back to the deck management page.

Alternative Flows:
2a. Player reached max decks.

1. System shows a message to the player that he cannot create a new deck because he has reached the maximum number of decks.
2. System suggests removing an existing deck or buying additional deck space.

4-5a. Player moved a card by accident or wants to move another card out of the deck.

1. Player selects the card in the deck he wants to move out of the deck.

2. The card moves out of his deck and the number of cards left he can put in his deck gets updated.

8a. Player wants to save without having selected enough cards.
1. System shows a message to the player that indicates how many more cards he has to select.
2. Deck is not saved into the database.

8b. Validation fails because the player already has a deck with the same name.
1. System shows a message to the player that he needs to choose a different name.
2. System highlights the input field for the deck name.

8c. Validation fails because the player didn't choose a deck name.
1. System shows a message to the player that a deck name is required.
2. System highlights the input field for the deck name.

8d. Validation fails because the player tried to manipulate the game by inserting cards he does not own.
1. System shows a message to the player that he has used cards which he does not own.
2. System removes the cards he does not own from the deck.

9a. Deck cannot be saved into the database.
1. System logs failed attempt to save into the database with all available error information.
2. System shows a message to the player that the deck could not be saved, and he should either try again or contact the support.

Special Requirements:
UI needs to be intuitive and work on both large (Desktop) and small (Mobile) screens.

Technology and Data Variations List:
Name is entered and saved in UTF-8

Frequency of Occurrence:
Could be the whole time with multiple players simultaneously.

Miscellaneous:
Open issues:
1. Define the number of cards in a deck

USE CASE 5: MANAGE DECKS

Main Success Scenario:
The user arrives at the deck management page and is presented with a list of his created decks. The user chooses a deck and the action (edit or delete) he wishes to execute. The system then decides whether to redirect the user to the create deck page and update his chosen deck with the changes made on that page or if the chosen deck should be deleted from the players account based on the action the user chose. After successful execution of the action chosen, the system redirects the user back to the deck management page.
Alternate Scenarios:
If the system fails to update the chosen deck with the changes made, the user gets redirected to the deck management page and an error is presented to him.
If the system fails to delete the chosen deck, the user gets redirected to the deck management page and an error is presented to him.
If the user submitted an invalid deck modification, the system redirects the user to the create deck page with the selected deck. The system displays an error to the user containing a list of the changes necessary to make the modification valid.

USE CASE 6: BUY DECK SPACE

Main Success Scenario:
A player wanting to extend his deck space can buy additional space on the deck management page by clicking on a button. He will then be presented a form which allows him to select how much more space he wants to buy and with which payment method. After selection and confirmation of the purchase, he will get redirected to a payment service where he will authorize the payment. The player will get redirected back to the homepage where the payment will be verified and in case of a successful payment, his deck space will get expanded by the amount he has payed for.
Alternate Scenarios:
If the player cancels the payment while he is on the page of the external payment provider, he will get redirected back to our page and will be shown a message informing him that the payment was not successful and the deck space will not get expanded.

USE CASE 7: SEARCH OPPONENT

Main success scenario:
The user can press the play button when he is logged in. The player is then added to a waiting pool. The system now compares the skill value of the user to the other players in the waiting pool. The system chooses an opponent with a skill value as similar as possible to the player or a skill value in a certain range. The system starts the game with the two matched players.

Alternate scenario:
While the user is waiting for a fitting match a waiting message is displayed. The searching process can be stopped with a well visible button in the UI.

USE CASE 8: PLAY ROUND

Scope: Battle Beasts - Browser Game
Level: User-goal
Primary Actor: Player
Stakeholders and Interests:

- Player: Wants a well working turn-based card game.
- Company: Wants to offer a well working application to maintain a big player base.

Preconditions:
Player is logged in and authenticated.
Postconditions:
Player receives a new card.
Player's skill value is updated.
Main success scenario:

1. The player starts a round with a chosen deck.
2. The player is matched against an opponent with a similar skill value.
3. The System randomly decides which player starts.
4. The player begins his first turn consisting of five phases.
5. The player begins the "start phase" in which the system fills up the players action points and adds new cards to his hand.
6. The player goes into the "cast phase" in which the player has the

possibility to cast animals from his hand onto the game's board. The player can cast as many animals as long as enough action points are available.
7. The player goes into the "spell phase" in which the player has the possibility to activate spell or equipment cards on his animals. The player can activate as many spell and equipment cards as long as enough action points are available.
8. The player goes into the "attack phase" in which the player has the possibility to attack the opponent's animals with his own animals or attack the opponent directly if no animals are on the opponents site.
9. The player goes into the "end phase" in which the system updates values and starts the turn of the opponent.

As soon as one of the players life points reaches zero the system ends the round and determines the winner.

Alternative Flows:

6a. The Player wants to cast or activate card but has not enough action points for the card.

1. A message is displayed informing the player that not enough action points are available.

Special Requirements:

UI needs to be intuitive and work on both large (Desktop) and small (Mobile) screens.

Technology and Data Variations List:

Cards and their initial attributes are stored in a JSON file.

Frequency of Occurrence:

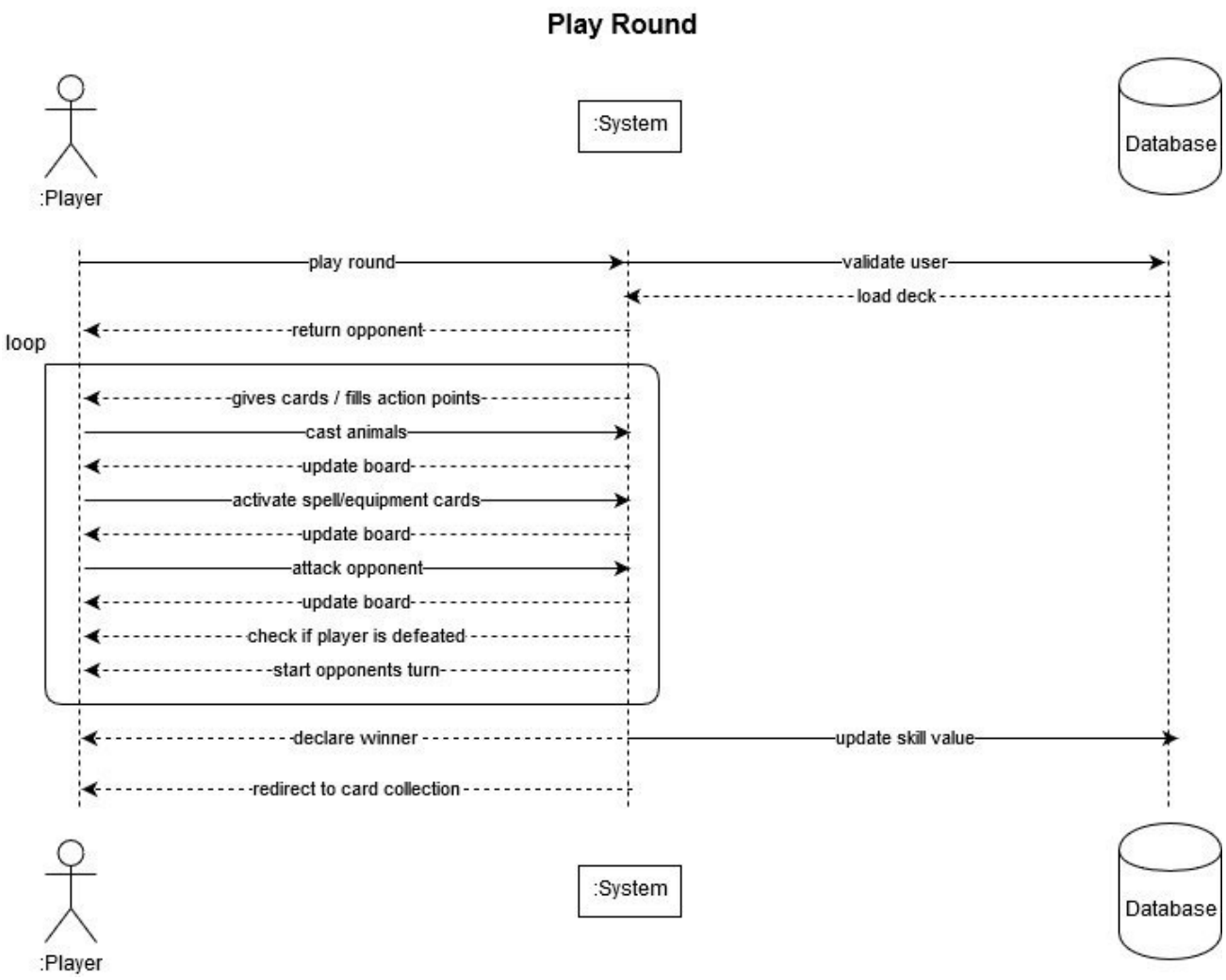
The whole time with multiple players simultaneously.

Miscellaneous:

Open issues:

1. Connection problems or player quitting during an active game.

USE CASE 8: PLAY ROUND



USE CASE 9: COLLECT CARD

After finishing a round of Battle Beasts the player receives a card. The card will be randomly selected from the system. The card will then be added to his personal Card collection and can be used in upcoming rounds.

USE CASE 10: BUY CARD

Main Success Scenario:
The user is logged in and navigates to the collection page. The user chooses the number of cards packs to purchase. The customer makes a purchase and is redirected to a payment service where the total is displayed. The customer enters the payment information and confirms the purchase. The system validates the payment information and handles the payment. The System saves the sale, updates the user's account and the sale record. The contents of the purchase are now on show in the user's card collection.
Alternate Scenario:
At any time the user can cancel his purchase. The systems aborts the purchase if the payment validation fails and displays a message to the user displaying that the purchase has failed and the payment information has to be reentered.

USE CASE DIAGRAM

ADDITIONAL REQUIREMENTS

Functional
The application consists of multiple parts:
- A web application for standard functionalities such as register, login and log out.
- A management part for the decks that are used in the game.
- A purchase area where cards or deck space can be bought.
- The main game
    - An automatic opponent search
    - The turn based card game
    - A reward card at the end of a game

All application parts are described more detailed in the use cases.

Usability
Through a very user friendly UI the web application is self explaining and multiple labeled buttons guide through the menus.
For the gameplay itself a manual will be available where every step is described in detail.
The basic rules of the game can be found under game description.

Reliability
All application parts have detailed tests in the frontend and backend that ensure correct inputs and updates of the user's data. This security measure is respected so that no data is lost in the process of a registration or payment.
Connection failures can occur during a game depending on the internet connection of the players. In such a case the last state of the game is saved and in case of a reconnection loaded again.

Performance
The application will be available all the time and only offline for small updates.

Supportability
The whole application is split into a frontend and a backend part. The files are structured in a detailed and specifically named folder structure. Through the use of typescript classes and react, the application is clearly readable and which makes it easier for programmers to understand the code.

ADDITIONAL REQUIREMENTS

All naming conventions were respected and the documentation is strictly written in english, enabling an international use.

Implementation
The resources needed for the application is an internet capable device such as a computer or mobile phone with a browser and a stable internet connection.
The game is released completely in english.

Interface
No external systems are needed for the application.

Operations
The game is hosted on a server and all data is stored in a database.

Packaging
The application will be released only online as a web application.

Legal
The application and all designs in it are property of the team behind Battle Beasts and the contractor of this project.

GAME RULES

Basics:

The idea is a turn-based card game where two players with a certain amount of health and action points battle each other until one party has no more health points left. The card game itself consists of animal cards which fight each other or attack the opponent directly, spell cards which enable the player to boost their animals or decrease the opponent's animals' skills temporarily and equipment cards which improve animals permanently. Every card costs a certain amount of "action points" so that you have a limitation of tactical moves for any turn you make.

Players and Cards:

The player has a deck of 20 cards, a set amount of action points and a certain amount of life points.

An animal card has a species, an action point value, an attack value and a defense value. Animals can be played in attack or defense mode. Only animals in attack mode can attack other animals.

A spell card has an effect and an action point value. If the card is played the effect will be activated instantly.

An equipment card has an effect and an action point value. If the card is played the effect lasts permanently on the chosen animal.

Action points are used to limit the players possibilities. Every card has an action point value and by the cast or activation of the card this action point value will be subtracted from the players action point values.

The game:

The game starts with two players drawing five cards from their deck and it will be randomly selected which player starts. The player which will be selected then starts his first turn.

A turn is split into five phases:

1. The start phase
   In the start phase the player draws a card from his deck and his action points will be filled up.
2. The cast phase
   In the cast phase the player can cast animals in attack or defense mode onto the field as long as there are enough action points available.
3. The spell phase
   In the spell phase the player can activate spell or equipment cards to upgrade his animals as long as there are enough action points available.
4. The attack phase
   In the attack phase the player can attack the enemy's animals or if the opponent's side is empty, the opponent directly.
5. The end phase
   In the end phase the life points are counted and killed animals are removed from the field.

GAME RULES

Fighting

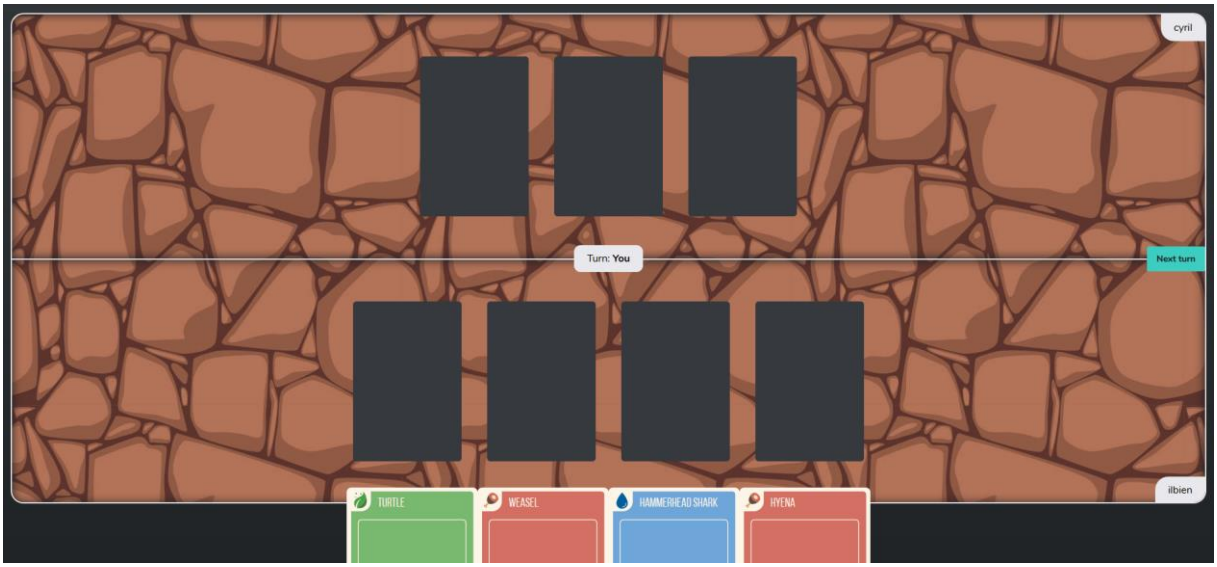
Only animals in attack mode can attack other animals. To attack and destroy another animal the attack points of the attacking animal need to be higher than the attack or defense value of the targeted animal. If an animal in defense mode is destroyed it will be removed from the field. If an animal in attack mode is destroyed it will be removed from the field and the difference between the attack point of the attacking animal and the targeted animal will be subtracted from the opponent's health points.

If there are no animals on the field the player can be attacked directly. If an animal attacks a player directly the damage will be subtracted from the players health points.

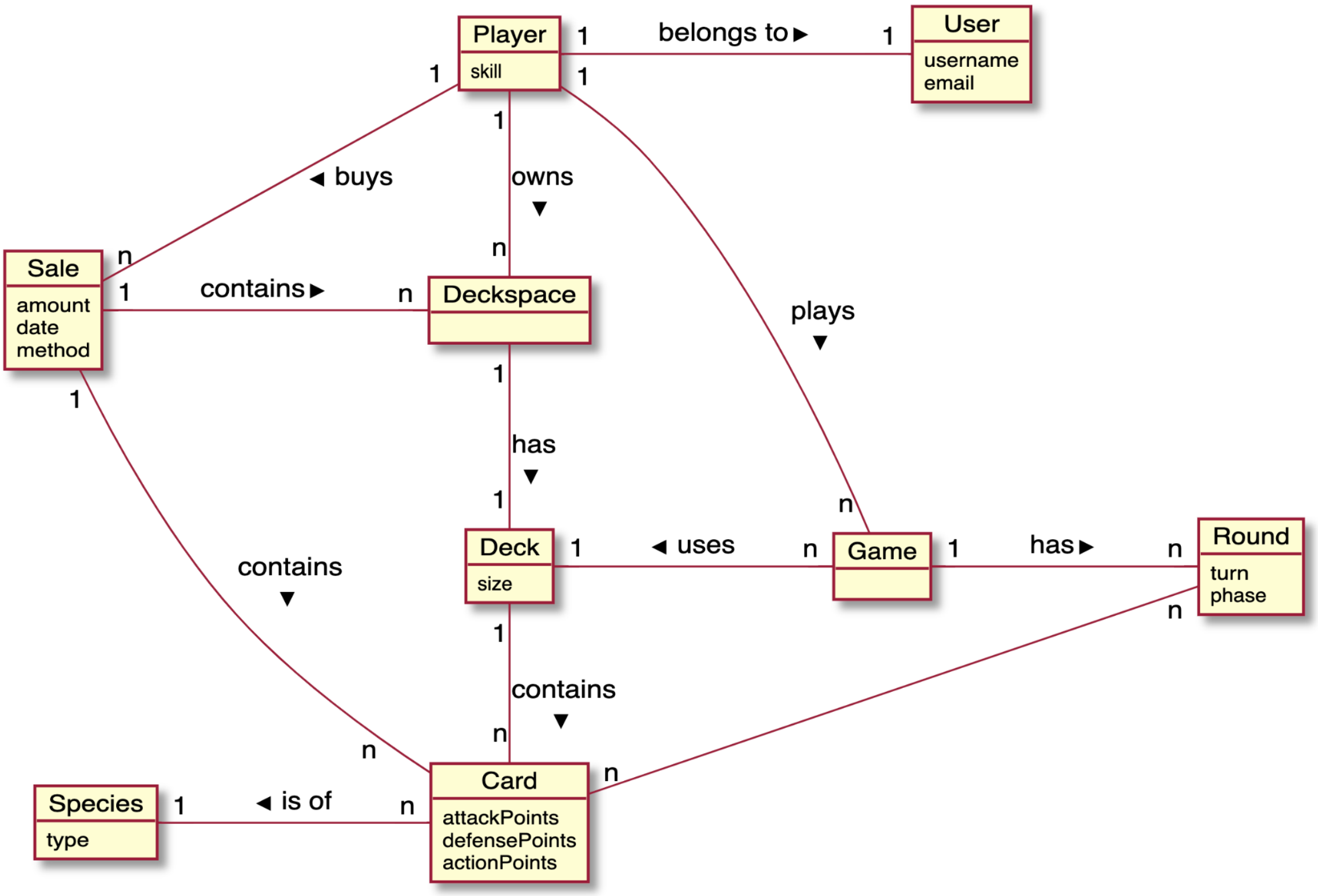
The field [1]

The field consists of as many animal cards a player will cast. The maximum will be set to ten to ensure a fair gameplay. Every animal card has a slot bellow itself where spell or equipment cards will be placed if activated. The hand is shown directly on the bottom of the screen.

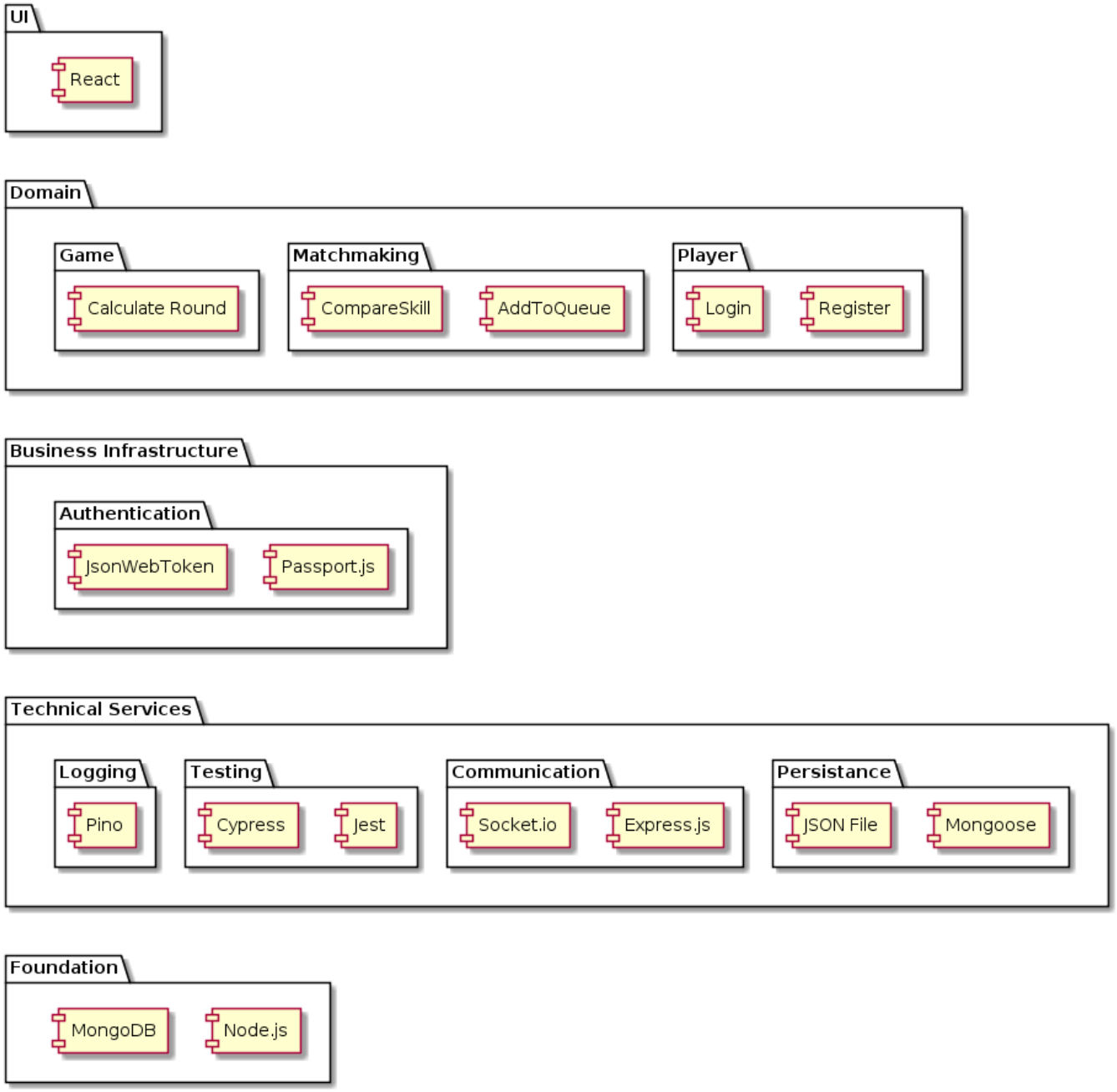
[1]



DOMAIN MODEL



SOFTWARE ARCHITECTURE



The software architecture for our project is based on the programming language Typescript which is implemented using Node.js in the backend and React in the frontend. This allows for fast, agile prototyping while also providing a future proof and stable solution in the long run. To match the asynchronous and event-driven nature of Node.js MongoDB is chosen as the database to provide persistence.

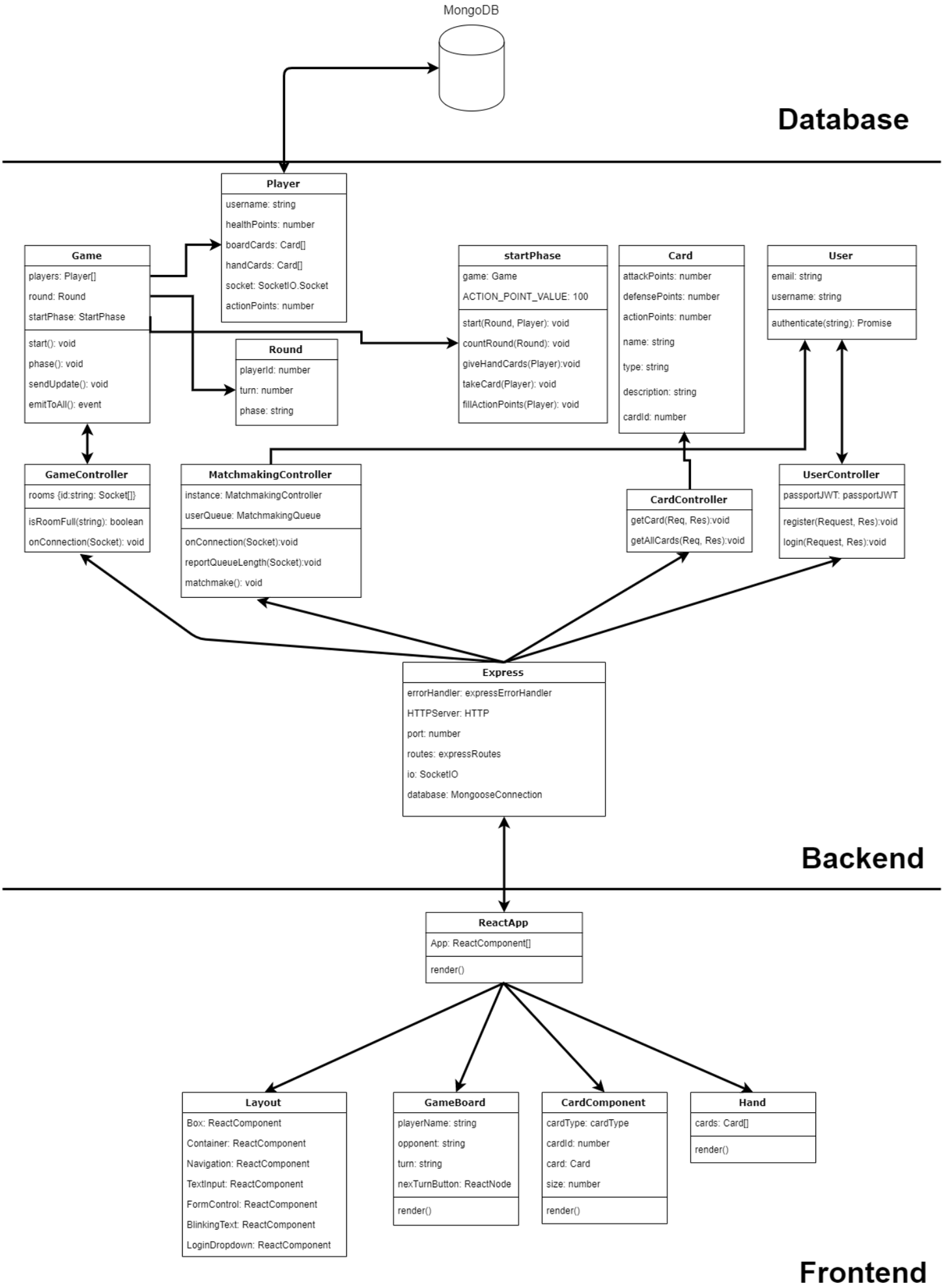
Regarding the basic technical services required, well tested, proven

and popular libraries are chosen. This includes Pino, the chosen logger, which is lightweight, fast and allows the output of both easily automatically consumable logs in the JSON format or pretty human readable text format.
For testing, Jest is used for unit testing and Cypress for End-To-End testing. Both these libraries are very well suited for the asynchronous nature of Typescript and have reached enough maturity to be used. To keep our code style consistent in the front- and backend and across developers we employ ESLint with the popular JavaScript Style Guide by AirBnb.
The communication between front- and backend in our product uses Socket.io for realtime Websocket based communication while a REST API interface implemented in Express.js is used for everything else like authentication. Socket.io was especially chosen because of it's solid handling of unstable connections, automatic reconnection and fallback to HTML polling should Websockets be unavailable on the used device and browser combination. Express.js was chosen because it represents the de-facto standard for REST API building in the Node.js ecosystem and is very customizable using middlewares.
Mongoose is used as an abstraction layer and ODM for MongoDB chosen for its ability to bring more structure and type safety to the document based storage system of MongoDB. Additionally to Mongoose, JSON files are used to store data which is rarely changed and the same for all users like game asset metadata.

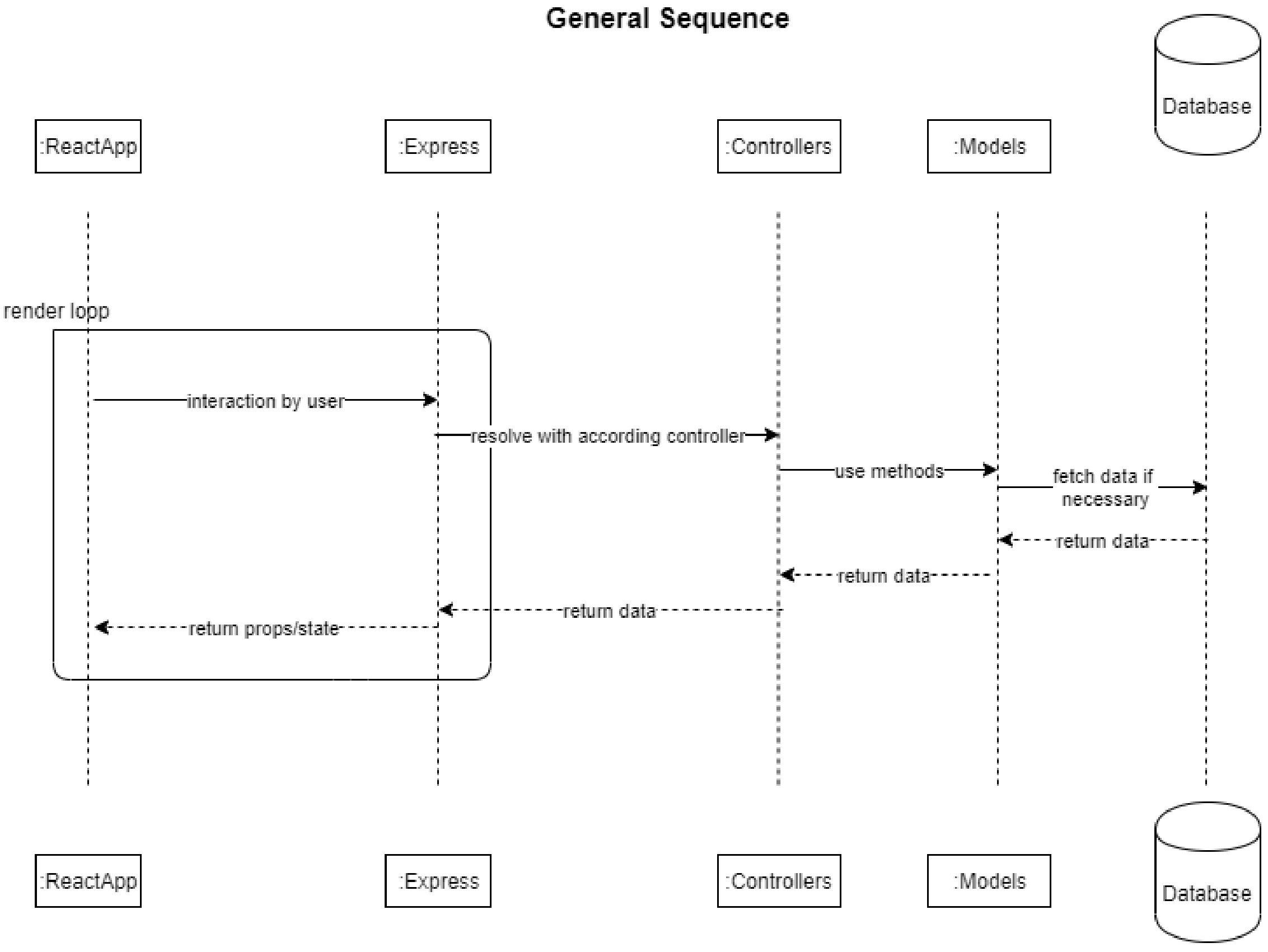
The business infrastructure of our product includes Authentication which is realised using the popular and very flexible library Passport.js. The authentication strategy used is JsonWebToken or JWT for short. This strategy is chosen because it's very well suited for single page applications that communicate with the backend using stateless API endpoints. It also removes the need of having to keep track of user sessions in the backend which in return increases and enables easy scalability.

Finally to create a responsive, fast and modern web application the widely used, company backed and proven React library is used. React was chosen because of its lightweight nature, amount of available documentation and existing experience in our development team.

DESIGN ARTIFACTS



DESIGN ARTIFACTS



IMPLEMENTATION

The validation of the software architecture was made during the first iteration of our project. We kept the following aspects in mind and compared them to the requirements of the project to make a decision:

1. Validation and clearance
2. System test
3. Integration test
4. Componenten-, module- respectively unit test
5. Development, programming

After discussing and weighing the arguments we finally had a rough idea of how we would implement the previous mentioned requirements:

1. Git (https://github.zhaw.ch/berchmar/PSIT3-HS19-IT18a_ZH-TeamIP1)
2. Requirement specification analysation
3. Interface requirements
4. Shallow, Enzyme
5. TypeScript, React, Node, MongoDB

INCEPTION PHASE

ITERATION 1

Task	Type	Description	Responsible	Estimated effort (in h)	Effective effort (in h)
1	Administrative	Idea finding	Cyril	16	16
2	Documents	Project sketch	Cyril	20	24
3	Administrative	Presentation	Marc	5	4
4	Documents	GUI Wire-frames	David	5	2
5	Documents	Architecture definition	Cyril	4	4
6	Documents	Analyze Use-case "Register account" and "Do login"	Cyril	2	1
7	Meeting	Update meeting (18.09.)	Cyril	7	8
8	Meeting	Update meeting (23.09.)	Cyril	5	5
9	Meeting	Update meeting (28.09.)	Cyril	7	9
10	Administrative	Participate in project presentations (30.09.)	Marc	15	15
				86	**88(+2)**

In the first iteration, the main goal was to find a project idea and write it down more detailed. We already decided on the architecture to be able to start with a prototype in the next iteration. In the end, we almost matched the estimated effort with only two hours too much.

ELABORATION PHASE

ITERATION 2

Task	Type	Description	Responsible	Estimated effort (in h)	Effective effort (in h)
1	Administrative	Development environment set up	Marc	6	5
2	Coding	Architectural prototype implemented	Marc	6	6
3	Coding	UC 1 “Register account”	Marc	10	14
4	Coding	UC 2 “Do login”	Cyril	10	6
5	Coding	UC 3 “Do logout”	Marcel	2	4
6	Documents	Prepare Use-cases for next iterations	Cyril	10	14
7	Meeting	Architecture workshop (04.10.)	Cyril	5	8
8	Learning	React & TypeScript Tutorials	Ilbien	10	8
9	Meeting	Update Meeting (07.10.)	Cyril	5	5
10	Meeting	Project Sketch feedback (07.10.)	Cyril	4	4
11	Meeting	Update Meeting (11.10.)	Cyril	5	5
12	Documents	Domain Model first version	Ilbien	5	5
13	Coding	Basic Design implementation	Cyril	2	2
14	Administrative	Update iteration plan	Cyril	1	1
15	Administrative	Prepare iteration 3	Cyril	3	3
16	Documents	Prepare example cards	David	4	4
17	Meeting	Update meeting (14.10.)	Cyril	5	5
				93	**99(+6)**

With iteration 2, we have implemented an architectural prototype that showcases all our different needs, for example, a RESTful API and real-time communication between two users. Additionally, we implemented some easier use-cases so the developers which are not familiar with the chosen technology have an easy start. Overall, we are six hours over the estimation because it took a bit more time to get on with our stack.

ELABORATION PHASE

ITERATION 3

Task	Type	Description	Responsible	Estimated effort (in h)	Effective effort (in h)
1	Documents	Finish domain model	Cyril	4	2
2	Coding	Implement dummy match making	Marc	3	6
3	Coding	UC 8 "Play round" - Create new game	Cyril	8	4
4	Coding	UC 8 "Play round" - Start phase	Ilbien	10	8
5	Coding	UC 8 "Play round" - Cast phase	Marcel	10	9
6	Coding	UC 8 "Play round" - Attack phase	Cyril	10	6
7	Coding	UC 8 "Play round" - End phase	Cyril	4	4
8	Coding	Store example cards in json file	David	4	7
9	Coding	UC 4 "Create deck"	Marc	12	10
10	Documents	Architecture Documentation - Use Cases	Ilbien	5	3
11	Documents	Architecture Documentation - Additional requirements	Ilbien	3	2
12	Documents	Architecture Documentation - Software architecture	Marc	2	2
13	Documents	Architecture Documentation - Design artifacts	David	2	3
14	Documents	Architecture Documentation - Implementation	Marcel	2	2
15	Documents	Architecture Documentation - Project management	Cyril	2	2
16	Administrative	Architecture Documentation - Review	Cyril	8	7
17	Documents	Architecture Documentation - Composition	David	2	2
18	Coding	Refactoring	Cyril	2	4
19	Administrative	Prepare presentation	Cyril	3	2
20	Meeting	Update meeting (18.10.)	Cyril	3	4
21	Meeting	Update meeting (21.10.)	Cyril	3	3
22	Meeting	Update meeting (25.10.)	Cyril	3	6
				105	**98(-7)**

ELABORATION PHASE

ITERATION 3

In iteration 3, we implemented a prototype of our main use-case “Play round” to prove that the main goal of our application is doable. Because we prepared the setup really well in the previous iterations, we ended up spending a bit less time than expected.
Because the prototype works well and as expected, we don’t have to make adjustments to our setup, features list or upcoming roadmap.

CONSTRUCTION PHASE

ITERATION 4

Task	Type	Description	Responsible	Estimated effort (in h)
1	Coding	UC 8 "Play round" - Spell phase	Cyril	20
2	Coding	UC 8 "Play round" - Improvements	Cyril	20
3	Coding	UC 7 "Search opponent" - Calculate player skill	Marcel	8
4	Coding	UC 7 "Search opponent" - Matchmaking	Marc	15
5	Coding	UC 5 "Manage decks"	David	10
6	Coding	Detect disconnected players	Ilbien	8
7	Meeting	Update meeting (28.10.)	Cyril	5
8	Meeting	Update meeting (1.11.)	Cyril	3
9	Meeting	Update meeting (4.11.)	Cyril	3
10	Meeting	Update meeting (8.11.)	Cyril	3
				95

In iteration 4, we will finish the main use-case “Play round” and improve the prototype implemented in iteration 3. Additionally, players will be matched accordingly to their skill to make the matches fairer.

CONSTRUCTION PHASE

ITERATION 5 (ROUGH SCHEDULE)

Task	Type	Description
1	Coding	UC 10 “Buy card”
2	Coding	UC 9 “Collect card”
3	Documents	First version of users manual
4	Meeting	Meetings

In iteration 5, we plan to implement two use-cases and already start with a first draft of the users manual.

ITERATION 6 (ROUGH SCHEDULE)

Task	Type	Description
1	Coding	UC 6 “Buy deck space”
2	Documents	Users manual finished
3	Documents	Final report
4	Administrative	Presentation
5	Coding	Testing, Bugfixing, Improvements
6	Meeting	Meetings

Iteration 6 marks the end of the project and a beta version should be ready. So we finish all use-cases, documents and do some testing and final improvements to provide a fully functioning beta version.

CONSTRUCTION PHASE

ITERATION 5 (ROUGH SCHEDULE)

Task	Type	Description
1	Coding	UC 10 “Buy card”
2	Coding	UC 9 “Collect card”
3	Documents	First version of users manual
4	Meeting	Meetings

In iteration 5, we plan to implement two use-cases and already start with a first draft of the users manual.

ITERATION 6 (ROUGH SCHEDULE)

Task	Type	Description
1	Coding	UC 6 “Buy deck space”
2	Documents	Users manual finished
3	Documents	Final report
4	Administrative	Presentation
5	Coding	Testing, Bugfixing, Improvements
6	Meeting	Meetings

Iteration 6 marks the end of the project and a beta version should be ready. So we finish all use-cases, documents and do some testing and final improvements to provide a fully functioning beta version.

RISKS

#	Risk	Probability	Damage Potential	Priority	Counter-measures
1	**Playerbase** Players who play similar games have invested time and money to collect lots of cards which leads to prestige among the players. This means people playing similar games won't be easy to fetch and there is a chance that our estimation is wrong.	High	High	1	Spend more on advertising the game on social media.
2	**Network stability** We require a stable network. Metcalfe's law points out that the benefit of a network grows quadratic as the number of users increases.	Medium	Medium	2	Use technologies capable of auto-scaling and run stress tests.
3	**Knowledge** Not every developer is already familiar with the technologies in use.	Medium	Low	3	Hold workshop-like meetings or do pair-programming when problems occur.
4	**Real-time protocol** There is a chance that the protocol used for real-time communication is not suited for games	Low	Low	4	Make sure the communication is implemented like in the prototype which has proven that it works.

GLOSSARY

Board	Place of the game where cards can be placed. All cards on the board are visible to both players.
Deck	A deck is a pool of cards that can be used for a game. Each player can create his own decks so he has more control over which cards appear during the game.
Hand cards	Specifies all cards which the player has on his hand. The opponent cannot see these cards.
JSON	JSON is an open-standard data format and easy to use in combination with JavaScript.
MongoDB	Is a document-based database which can store JSON-like documents.
RESTful API	A RESTful API is an application program interface that uses HTTP requests to GET, PUT, POST and DELETE data.
Skill level	Indicates the skill level of a player. After each game, his skill will get calculated depending on the skill of the opponent and the outcome of the game.
WebSocket	Is a real-time communication protocol for the web and used to send changes in the game in real-time to the server and the opponent.