BATTLE BEASTS
Project vision

PSIT4 | IT18a_ZH

Aleksandra Timofeeva
Cyril Wanner
Erwin Tran
Lars Höhener
Ilbien Paul
Marc Berchtold
Marcel Brennwald
Marius Niklaus
Michael Schlaubitz

Version 1.1 09.03.2020

Table of versions

Version	Date	Changes
1.0	03.03.2020	Initial Version
1.1	09.03.2020	Project vision document finished.

Table 1: Version history

Table of contents

1. Context

Gaming is more popular than ever in the entertainment industry and is expected to keep growing. Today there are various different computer games on the market. It’s getting harder to acquire the customers’ interest. Games connected to real world facts are of particular interest for many players since they can gain new useful knowledge in a playful way. Therefore, the best idea is to combine digital gaming with the theme of real-world animals.

Battle easts is a web-application in the form of a turn-based card game for two players. Each player has a certain amount of cards showing animals or equipment. Each animal card consists of a name, animal photo and short description. The cards have three different stats: the attack value, the defense value and the action points. Depending on these parameters, the player can win or lose. Moreover each player has health points which the player gains or loses during the game. To improve the points, players can also use special equipment cards.

To start the game, the user must register. However, the biggest difference and advantage among competitors is that this game can be played directly in the browser on mobile phones and computers.

This game is developed for players of any age. Colorfully designed cards and easy-to-understand rules will immediately obtain users’ interest. The actors are player, product owner, developer, supporter.

2. Objectives and functional overview

Battle Beasts is an online card game with already existing functions, such as registering an account, assembling decks, a working matchmaking system and the possibility to play multiple rounds against real enemies.
In this project the main objectives are creating additional features for the game, implementing third party functions and optimizing already existing features. The planned objectives can be put into categories and are rated by importance:

- 1 – Not important / optional
- 2 – Should be implemented / gives an advantage
- 3 – Important / needed

Visual design:
The game will be optimized with more visuals and animations. It will be possible to see the opponent's moves (BEAST-3), attack animations for the cards will be implemented (BEAST-4) and attack indicators will be included (BEAST-5) which ensure that the player can directly recognize which cards are able to attack and which not.

Furthermore, the deck creator will receive a button to directly buy new cards (BEAST-32). A drag and drop function will be added for the deck creator (BEAST-33). Deck names will receive a new design (BEAST-35). Cards will receive different colors and pictures in the deck making list (BEAST-38).
Importance: 2 With better visualization and animations the game will be better to market.

Game modes:
It will be possible for the player to choose between game modes such as casual and ranked (BEAST-10). Depending on the players win rate in certain modes the player will get a player rank (BEAST-12) and will be shown on a leaderboard (BEAST-11)
Importance: 3 Competitive players which are the main audience of this game will be hooked faster and play more rounds to increase their player ranks.

Opponent AI:
The game will feature an AI to play against to train before playing against real opponents (BEAST-13). The player can choose between different difficulties of the AI to optimize the training effect (BEAST-15).
Importance: 3 Players which have few experiences in card games will have a better introduction into the game.

Game content:
The game will be expanded with more animal cards (BEAST-16), spell cards (BEAST-18) and a new function to attack an opponent's card with multiple cards (BEAST-19).
Importance: 2 More content and a fair balancing will attract more players.

Mobile friendly
The game will receive a mobile friendly design (BEAST-25), a new game board (BEAST-22) and the cards will be optimized for mobile screens (BEAST-24).
Importance: 3 The main objective is to make the game playable on the phone to reach a bigger player base.

Payment:
The game will receive a third party payment system (BEAST-30) which will give players the opportunity to buy cards (BEAST-27) and deck space (BEAST-29).
Importance: 3 This will create the biggest part of the income.

3. Quality Attributes / non-functional requirements

Usability Through a very user friendly UI the web application is self-explanatory and multiple labelled 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. In the game itself animations and visualizations make every move clearly visible for the player.

Reliability / Security 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 implemented so that no data is lost in a registration or payment process. Connection failures can occur during a game depending on the internet connection of the player. In such a case the last state of the game is saved and in case of a reconnection loaded again.

Availability The application will be available online at all times with the exception of update down times, which will last two to three hours.

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.
In section five more coding conventions can be found.
All naming conventions were respected and the documentation is strictly written in English enabling an international use.

Implementation The resources needed for the application are 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.

Packaging The application will be released online as a web application and as a mobile web application.

Legal The application and all designs in it are property of the team behind Battle Beasts and the contractor of this project. All pictures and logos are not copyright claimed or created by our team.

Internationalization All descriptions and menus in the game are written in English to gain an international audience.

4. Constraints

#	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 estimated player base is wrong.	High	High	1	Spend more on advertising the game on social media.
2	**Network stability** BattleBeasts requires a stable network. Metcal-fe's law points out that the benefit of a network grows quadratic as the num- ber of users increases.	Medium	Medium	2	Use technologies capable of auto-scaling and run stress tests.
3	**Knowledge** Not every developer is 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.

Table 2: Constraints

5. Principles

The following principles and practices should be read and understood by all team members. Applying them successfully will allow us to develop better software in many regards.

Follow these Clean Code principles:

- Don’t Repeat Yourself (DRY)
- Keep it simple, stupid (KISS)
- Beware of Optimizations!
- Favour Composition over Inheritance (FCoI)
- Integration Operation Segregation Principle (IOSP)

- Single Level of Abstraction (SLA)
- Single Responsibility Principle (SRP)
- Separation of Concerns (SoC)
- Source Code Conventions
  - Style guide for both the frontend and backend: Airbnb JavaScript Style Guide.
  - Additionally, the Airbnb React/JSX Style Guide for the frontend.

- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)
- Liskov Substitution Principle
- Principle of Least Astonishment
- Information Hiding Principle

- Open Closed Principle
- Tell, don't ask
- Law of Demeter

Source: Clean Code Developer

6. 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 the team employs ESLint with the popular JavaScript Style Guide by AirBnb.
The communication between front- and backend in our product uses Socket.io for real-time 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 its 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 realized 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.

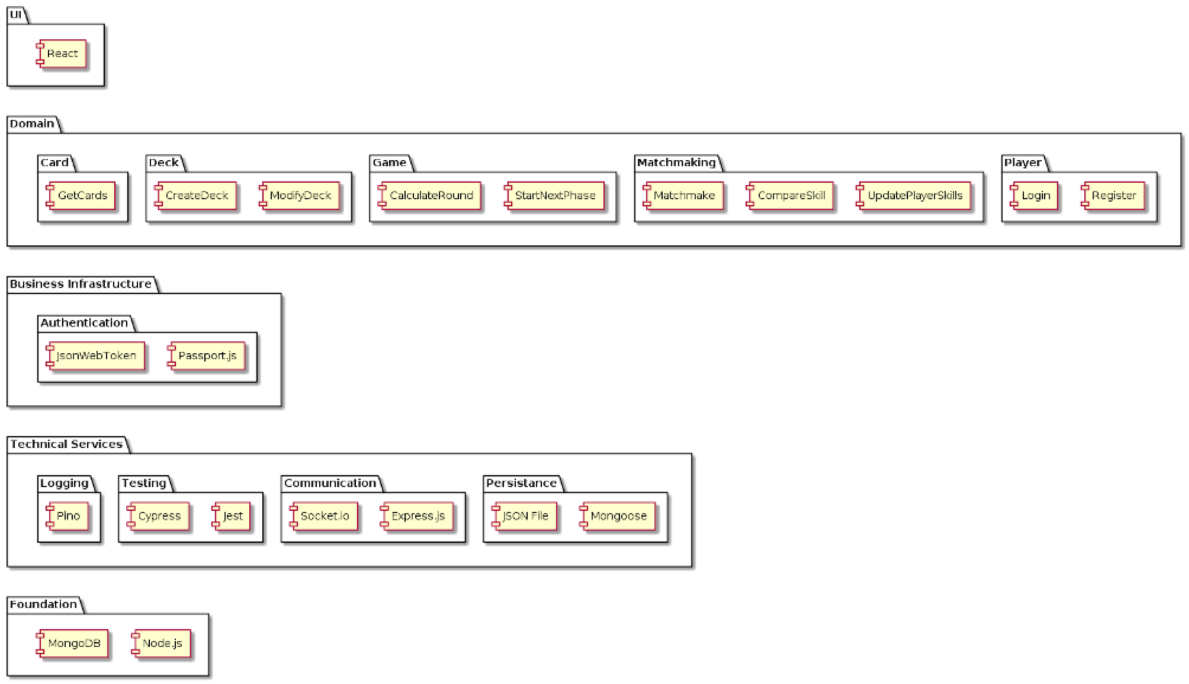

Figure 1: Architecture diagram

7. External Interfaces

The only external interface which will be used in Battle Beasts is the PayPal-API. By adding microtransaction to Battle Beasts, the team needs to include a payment service. The decision was made to use PayPal because of an easy to use and accessible API.

The communication of the Battle Beasts application takes place via a REST interface of the PayPal API. By including an API token in our requests, the application of Battle Beasts can communicate with the PayPal developer service.
During development, the team will be using a sandbox account in PayPal. Therefore, no real money needs to be transferred during development and testing.

8. Code

A key part of our application is the communication between the frontend and backend as Battle Beasts uses two different communication interfaces: a RESTful API and a realtime WebSocket connection. The RESTful API is used where realtime data is not mandatory, such as user registration, login or deck management calls. But when a user is starting a new game, a new WebSocket connection gets established as described in figure 2.

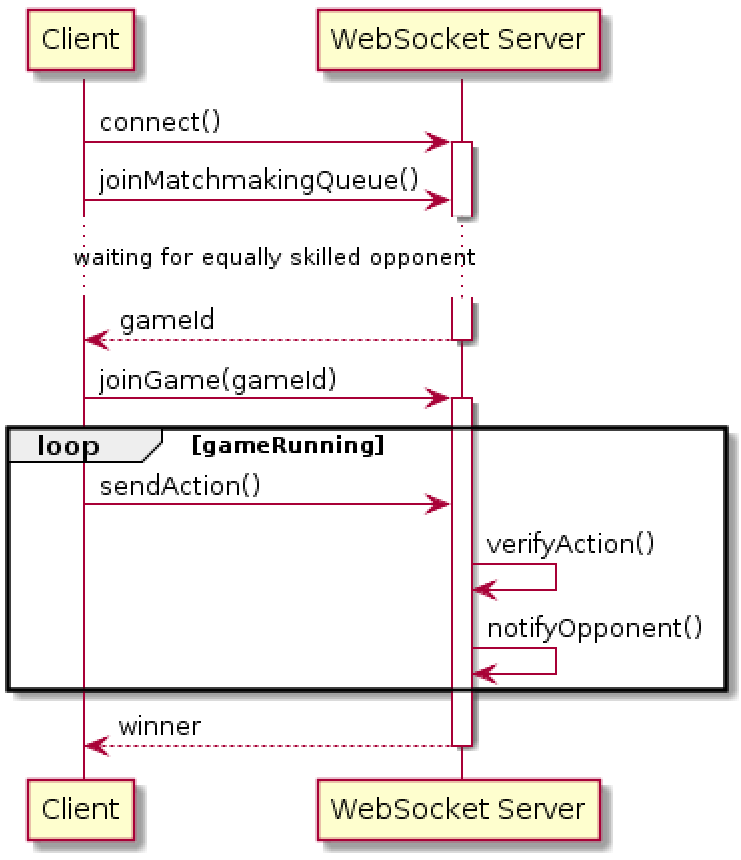

Figure 2: Game creation

After the WebSocket connection has been established, the player connects to the matchmaking room in which he waits to be assigned to a game room. As soon as the server finds two equally skilled players, he sends both of them to a newly created game room and the players leave the matchmaking room. Within this game room, the players and the server exchange their game states according to the actions the players executed.
On the client side, the game state is kept in the React state which automatically triggers a re-render of the changed parts each time an update gets received from the server.

9. Data

The user data is handled in the backend with the use of MongoDB, which follows a document-oriented model, rather than a traditional relational model. This means that game data is stored in JSON-like documents in the database. In combination with MongoDB the team uses Mongoose as an ODM which allows us to seamlessly retrieve and store data into the database.
Currently our application stores the players information and their created decks and owned cards.

10. Infrastructure

The software and MongoDB instance is being run in a Docker container on a Linux VPS. This setup enables easy deployment and scaling should it be needed, by using the builtin Docker Swarm load balancer. Redundancy could easily be added using multiple Docker Swarm nodes.

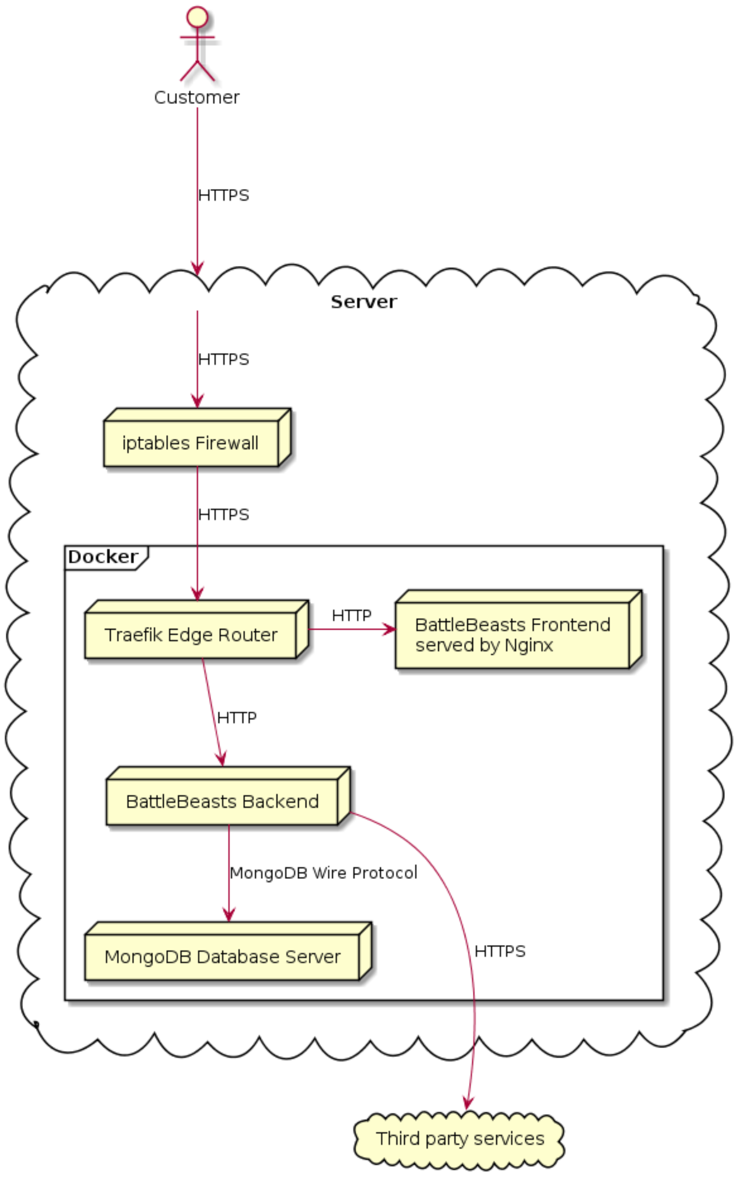

Figure 3: Infrastructure diagram

11. Deployment

The software is deployed to our private server using Docker Compose with continuous integration (CI). The compose file builds all of the necessary Docker images and the CI deploys them on the server if the tests run successfully.
Since everything is run in Docker containers there is no need to install anything other than the docker container engine on the server. This is done using apt:
sudo apt install docker

12. Operation and Support

To monitor the quality of our software SonarQube is used to evaluate test results and inform the team of Battle Beasts if the quality of the tests is not up to our standards.

To log errors and exceptions the Pino framework, a Node.js logger, is used.

13. Decision log

Domain	Outcome	Reason
Frontend-framework	React with TypeScript	React is one of the leading frontend frameworks and backed by multiple big companies, which allows us to write a future-proof and stable application. Since it has a large community there are a lot of integrations available and help can be found easily. The addition of TypeScript allows us to write type-safe code and errors can be easily detected while writing the code.
Backend-framework	Node.js & Express with TypeScript	The backend technology should be as close to the frontend technology as it allows developers to easily work on both parts and code can also be shared between them. This combination fulfills our requirements and is also very popular in the industry. It can also scale well and is easy to integrate in a serverless environment.
Database	MongoDB	As our product is not database heavy a lightweight solution was preferred. MongoDB fits perfectly, as it is a schema-less database and is integrated well in the Node.js ecosystem.
Git hosting	github.com	The team decided to move from the GitHub Enterprise hosting by ZHAW to the public GitHub as it offers more features, such as the new integrated CI (GitHub Actions).
CI	GitHub Actions	The team didn’t want to use yet another tool for our CI service so the one already integrated in GitHub was used.

Project management software	JIRA Cloud	After an evaluation of GitHub Issues, Trello and JIRA, the team decided to go for JIRA Cloud which is totally free for up to 10 users. It offers some useful features which the other products didn’t provide, such as time logging, estimations, sprints, epics and detailed customization possibilities.
Coding style guide	AirBnb	The JavaScript Style Guide by AirBnb is one of the most popular in the JavaScript community and offers many well thought-out decisions which are all documented in detail, so it is easy to understand why a certain rule must be followed. The extension for React also checks for common mistakes regarding performance and accessibility.
WebSocket framework	Socket.io	After a small prototype Socket.io met all our expectations in performance, simplicity and documentation. As it is one of the most popular solutions and used for many websites, it can be assumed that it is really robust and works even with a lot of traffic.

Table 3: Decision log

14. Table of figures

15. Table of tables