BattleSprout

Web-Anwendungsentwicklung Sommersemester 2023

Petrov, Ilia
Medieninformatik
i.petrov@oth-aw.de

Klinger, Fabian
Medieninformatik
f.klinger1@oth-aw.de

Kasseckert, Tassilo
Medieninformatik
t.kasseckert@oth-aw.de

Schmidt, Fabian
Medieninformatik
f.schmidt3@oth-aw.de

Stricker, Natalie
Medieninformatik
n.stricker@oth-aw.de

Kietzer, Rebecca
Medieninformatik
r.kietzer@oth-aw.de

Barbee, Montell
Medieninformatik
m.barbee@oth-aw.de

***Abstract*— Dieser Technical Report beinhaltet eine Beschreibung der Software-Architektur des Projektes BattleSprout. BattleSprout ist eine “Schiffe versenken”-Anwendung, welche den Fokus auf einen positiven, wettbewerbsorientierten Kontext legt. Der technische Report konzentriert sich auf die technischen Aspekte und erläutert, wie mithilfe eines React-Frontends und eines hybriden Backends, bestehend aus Express-Endpunkten und Socket.IO-Socketverbindungen, ein skalierbares Multiplayer-Spiel umgesetzt werden kann.**

I. Einleitung

A. Mission Statement

Bei BattleSprout handelt es sich um eine Web-Anwendung, die es Spielern erlaubt, eine etwas andere Variation des bekannten Brettspiels "Schiffe versenken” zu spielen.

Der Fokus liegt hier auf der Kombination von Wettbewerb und Kooperation. Es geht darum, ein Strategiespiel zu erstellen, bei dem der Kern auf Zusammenarbeit liegt, jedoch trotzdem das Wettbewerbselement erhalten bleibt.

Dadurch soll beobachtet werden, ob sich die Spielermentalität ändert, wenn das Ziel ist, dem anderen Spieler zu helfen und nicht wie bei "Schiffe versenken”, die gegnerische Flotte zerstören zu müssen.

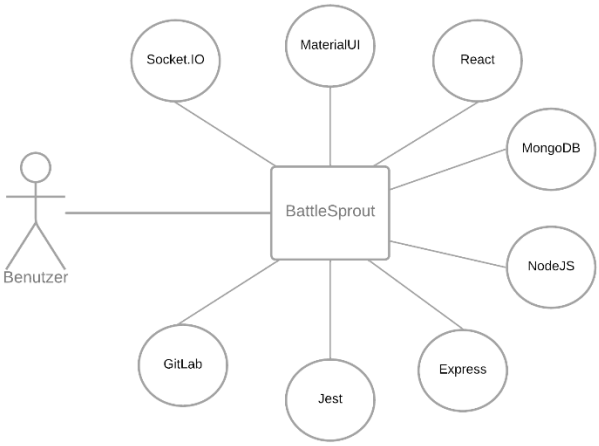
B. Spielregeln und Spielmechanik

Spieler müssen in BattleSprout strategisch vorgehen, um den Garten des Herausforderers zuerst erblühen zu lassen.

Vor dem Beginn des eigentlichen Spiels müssen die Spieler ihre Pflanzen auf einem Raster platzieren. Jeder Spieler besitzt ein eigenes Raster mit Zellen, auf denen er Pflanzen platzieren kann. Dazu kommen Wasserbomben, die abwechselnd von den Spielern auf die Zellen des Herausforderers geworfen werden müssen. Trifft man dabei eine Pflanze, so blüht diese auf. Der Spieler, der zuerst alle Pflanzen des anderen erblühen lässt, gewinnt das Spiel.

C. Kontextabgrenzung

Die Darstellung zeigt das System als Blackbox sowie die Fremdkomponenten und die Interaktion mit dem Benutzer.



D. Architekturziele

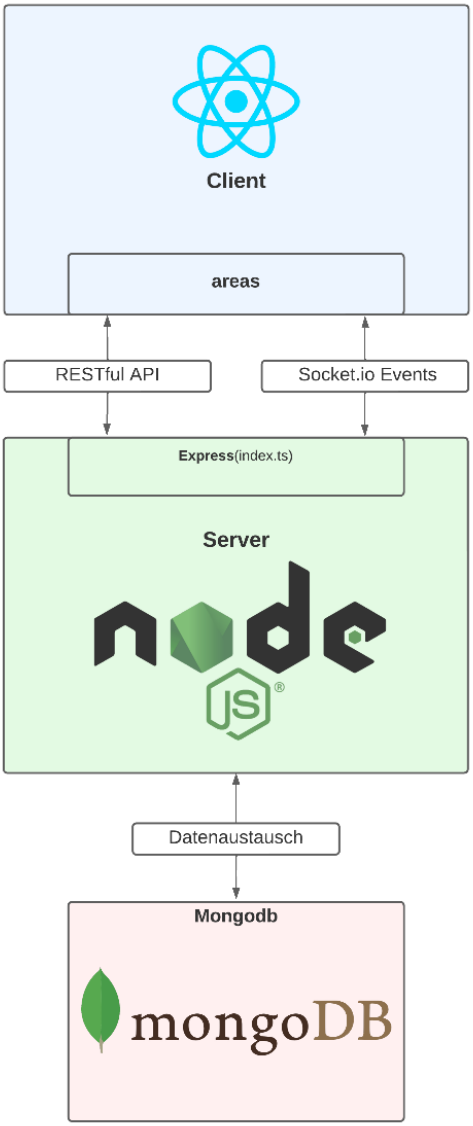
Das System soll folgende Architekturziele berücksichtigen:

- Zuverlässigkeit
  - Der Nutzer soll eine reibungslose Spielerfahrung haben
  - Die Verarbeitung der Spielaktionen und die Aktualisierung der Spielzustände müssen zuverlässig sein, um Spielausfälle zu vermeiden
- Benutzbarkeit
  - Die Anwendung soll für eine breite Gruppe von Nutzern intuitiv navigierbar sein
- Echtzeitkommunikation
  - Schnelle Aktualisierung des Spielstatus und der Darstellung der Spieleaktionen

II. Lösungsstrategie

A. Systemüberblick

BattleSprout verwendet den MERN-Stack mit Typescript als Programmiersprache, um die Vorteile der statischen Typisierung zu nutzen. Zusätzlich wird Socket.IO integriert, um eine Echtzeitkommunikation zwischen Client und Server zu ermöglichen.



B. Frontend Architektur

Im Frontend wird React mit Typescript als Programmiersprache verwendet. Durch die wiederverwendbaren Komponenten bei der Entwicklung mit React kann man verschiedene Elemente wie das Spielbrett, die Freundesliste oder den Login eigenständig behandeln. Dies ist vorteilhaft bei der Entwicklung mit mehreren Personen, da man so unabhängig voneinander arbeiten kann.

Bei Spielen ist es wichtig, den derzeitigen Zustand eines Spiels im Auge zu behalten und bei Änderungen zu aktualisieren. Die effiziente Zustandsverwaltung mit React funktioniert hierfür sehr gut.

Typescript ermöglicht es, Funktionen und Schnittstellen klar zu definieren und somit mögliche Verwirrungen bei der Eingabe und Ausgabe der Typen frühzeitig zu verhindern.

Die Hauptelemente der Architektur im Frontend teilen sich folgendermaßen auf:

1) Assets: Hier befinden sich die Bilder und Grafiken zur ansprechenderen Darstellung der Anwendung. Darunter fallen zum Beispiel das Hintergrundbild der Gesamtanwendung und verschiedene Pflanzenarten, um das visuelle Erlebnis der Nutzer zu verbessern, sowie das für BattleSprout angefertigte Logo.

2) CreateGame: In der CreateGame-Komponente kann der Nutzer ein neues Spiel erstellen. Dabei kann er Einstellungen wie zum Beispiel die Größe des Spielbretts wählen und das Spiel nach seinen Vorlieben anpassen.

3) FriendList: Die FriendList-Komponente ist im Layout der Seite eingebaut und zu jeder Zeit für den Nutzer über ein Burgermenü in der linken oberen Seite der Anwendung zugreifbar. Hier können Freunde hinzugefügt und entfernt werden.

4) GameField: Das GameField stellt den Hauptbereich der Anwendung dar. Es kann verschiedene Zustände haben, zum Beispiel den Zustand “setup” - in diesem Zustand wird nur das eigene Spielbrett dargestellt, auf dem dann die Pflanzen platziert werden können - oder den Zustand “playing” - in dem sowohl das eigene Spielbrett dargestellt wird, als auch das Spielbrett des gegnerischen Spielers, auf welches dann Wasserbomben geworfen werden, um die Pflanzen des Gegners zum Blühen zu bringen. Während des Zustands “playing” wird jeweils beiden Spielern angezeigt, wer gerade an der Reihe ist. Steht ein Gewinner fest, wird in den Zustand “finished” übergegangen und beiden Spielern signalisiert, dass das Spiel zu Ende ist.

Die Spielbretter werden mithilfe eines Gitters realisiert, die aus einzelnen Zellen bestehen. Die Zellen tragen jeweils ihre Position und eine Variable, um zu prüfen, ob sie schon bewässert worden sind. Für die Vorbereitungsphase ist außerdem eine Komponente definiert, die dem Spieler die Zahl der Pflanzen anzeigt, die noch platzieren müssen, bevor das Spiel starten kann.

5) Login und Signup: Die Login- und Signup-Komponenten erlauben es dem Nutzer, sich in die Anwendung einzuloggen oder ein neues Konto zu erstellen. Es ist wichtig anzumerken, dass nur eingeloggte und autorisierte Nutzer Zugriff auf alle Funktionen haben. Um auf die Hauptfunktionen der Anwendung, wie z.B. das Erstellen eines Spiels, die Freundesliste, das Spielbrett oder die Rangliste zugreifen zu können, muss der Nutzer vorher erfogleich eingeloggt sein.

6) Ranking: Die Ranking-Komponente zeigt die Rangliste der Spieler mit den meisten Siegen insgesamt an. Nutzer können sehen, wie sie im Vergleich zu anderen Personen stehen.

7) MainMenu: Die MainMenu Komponente ist der Startbildschirm der Anwenung, wenn der Nutzer angemeldet ist. Von hier aus kann er über den Druck einer Funktion zu Spiel erstellen, Spiel beitreten oder der Rangliste navigieren.

8) Layout: Hier ist die Oberfläche für die Obere Leiste und den Hilfedialog hinterlegt. Layout wird immer über den anderen Komponenten angezeigt, wenn der Nutzer angemeldet ist.

C. Backend Architektur

Für die Speicherung von Benutzerdaten, der Rangliste und der Freundesliste wurde MongoDB gewählt. MongoDB bietet uns die Flexibilität, Datenstrukturen anzupassen und zu erweitern, was in der Spieleentwicklung besonders wichtig ist. Da sich die Anforderungen im Laufe der Entwicklung ändern können, müssen Datenstrukturen an neue Gegebenheiten anpassbar sein. MongoDB erfüllt diese Anforderung souverän und ermöglicht eine effiziente Datenverwaltung.

Unsere HTTP-Endpunkte wurden mittels Node.js und dem Express-Framework umgesetzt. Dabei werden HTTP-Anfragen zu entsprechenden Controllern weitergeleitet und dort verarbeitet. Speziell sind `/login` und `/signup` Endpunkte, die über POST-Anfragen unser Identifikationsmodul implementieren.

Der Endpunkt `/newgame` ermöglicht das Erstellen eines Spiels durch eine POST-Anfrage mit dem Parameter `gameSize`. Das Spiel wird anschließend anhand seiner GameID in einer Map auf dem Backend-Server gespeichert. Diese Struktur ermöglicht es, das Spiel später effektiv über eine Socket-Verbindung zuzuordnen.

Für die Authentifizierung wird JWT (JSON Web Token) verwendet. Ein Token kann über den `/login` Endpunkt bezogen werden. Weiterhin kann durch das Socket-Ereignis `authenticate` und dem Token als Parameter eine Authentifizierung der Socket-Verbindung erreicht werden.

Die Verbindung zum Spiel geschieht über WebSockets. In diesem Kontext fiel die Wahl auf Socket.IO, da innerhalb des Teams bereits Kenntnisse darüber vorhanden waren.Der Backend-Server speichert den Spielzustand im Arbeitsspeicher (In-Memory). Über die WebSocket-Verbindung werden Zustandsänderungen in Echtzeit an die Clients übermittelt.

Ein wesentlicher Vorteil einer zustandsbehafteten WebSocket-Verbindung im Gegensatz zu einer zustandslosen Verbindung, wie beispielsweise HTTP, liegt in der Effizienz der Authentifizierung. Nachdem die Verbindung einmal authentifiziert wurde, können die Verbindungsinformationen zusammen mit der entsprechenden Spiel-ID und Spieler-ID gespeichert werden. Dies macht die Verbindung besonders leichtgewichtig.

Das Backend lässt sich in die folgenden Abschnitte einteilen:

1) Socket: Enthält die Implementierung der Echtzeitkommunikation zwischen dem Server und dem Client während eines Spiels. Hier werden die Validierung und die Handshakes durchgeführt, um sicherzustellen, dass nur autorisierte Clients auf die jeweiligen Socket-Verbindungen zugreifen können. Des Weiteren werden hier verschiedene Events definiert, die während eines Spiels auftreten können. Bei „handleSetSplash“ beispielsweise werden der Spieler, der die Aktion ausgeführt hat und die entsprechenden Koordinaten, die dieser gewählt hat, in eine Funktion übergeben, die die Logik der Aktion implementiert.

2) Models: Der Model-Ordner beinhaltet unsere definierten Datenstrukturen, wie zum Beispiel einen Nutzer, der eine E-Mail, ein Passwort und Freunde besitzt.

3) Routes: Enthält die API-Routen der Anwendung für die Kommunikation außerhalb eines Spiels. Dort sind zum Beispiel Routen für die Verwaltung von Freunden, der registrierten Nutzer und der Rangliste enthalten.

4) Game: Enthält die Spielelogik. Jedes Spiel hat genau eine Instanz von dem Objekt “Game”. In diesem Objekt wird der komplette Spielezustand gespeichert. Es ist aufgeteilt in Game als Fascade Controller , dem Plant Objekt, das die Logic für die Pflanzen übernimmt und dem PlantTile Objekt, das den Zustand von jeder Pflanzen Kachel speichert.

III. Probleme bei der Entwicklung

Durch die vielen einzelnen Komponenten, welche von verschiedenen Team-Mitgliedern erstellt wurden, wurde die Projektdatei schnell sehr groß, und die Gefahr, den Überblick über verteilte Funktionen und deren Zusammenhänge zu verlieren, größer. Deshalb war es wichtig, sich bei den Besprechungen immer gegenseitig mitzuteilen, an was man in der letzten Woche gearbeitet hatte und sich stetig einen Überblick über alle Komponenten zu verschaffen.

Manchmal waren die Anforderungen sehr komplex und das erforderte eine sorgfältige Planung und Umsetzung. Wenn die gewünschte Funktionalität schwierig zu implementieren war, oder Kenntnisse in speziellen Technologien (Bibliotheken, Frameworks) erforderte, führte dies zu Verzögerungen und technischen Herausforderungen. Das Problem haben wir gelöst durch Recherche und Lernen neuer Technologien, Zusammenarbeit im Team und regelmäßige Kommunikation miteinander

Aufgrund unterschiedlicher Erfahrungen der Teammitglieder wiesen viele Frontend-Komponenten unterschiedliche Darstellungsstile auf. Dadurch hatte das System ein uneinheitliches Erscheinungsbild, wobei alle funktionalen Elemente vorhanden waren. Um dieses Problem anzugehen, war es erforderlich, regelmäßige Überarbeitungen fast aller Komponenten durchzuführen, ohne die Arbeit anderer zu beeinträchtigen. Dabei orientierten wir uns am in Figma erstellten Prototypen. Um das Styling zu vereinfachen, nutzten wir MUI (Material UI) Komponenten an allen möglichen Stellen. Die Teammitglieder machten sich schnell mit der Dokumentation vertraut, um damit umzugehen.

Die Definition der Events innerhalb der Socket-Verbindung stellte eine Herausforderung dar. Um dieses Problem zu adressieren, wurden frühzeitig "Handler"-Objekte erstellt. Diese waren zunächst nicht implementiert, dienten aber dazu, Verträge zwischen dem Frontend und Backend festzulegen. Dies ermöglichte es dem Frontend, mit der Implementierung zu beginnen, ohne darauf warten zu müssen, dass das Backend alle Implementierungen abgeschlossen hatte.

IV. Zukunftsausblick

Derzeitig bietet BattleSprout eine Möglichkeit eine Variante des Spiels „Schiffe versenken“ zu spielen.

Darüber hinaus könnte eine Verbesserung der Benutzeroberfläche und des Designs vorgenommen werden, um die Benutzerfreundlichkeit zu erhöhen und eine ansprechendere visuelle Erfahrung zu bieten. Dies kann mit Hilfe der Einbindung von Animationen und Soundeffekten realisiert werden.

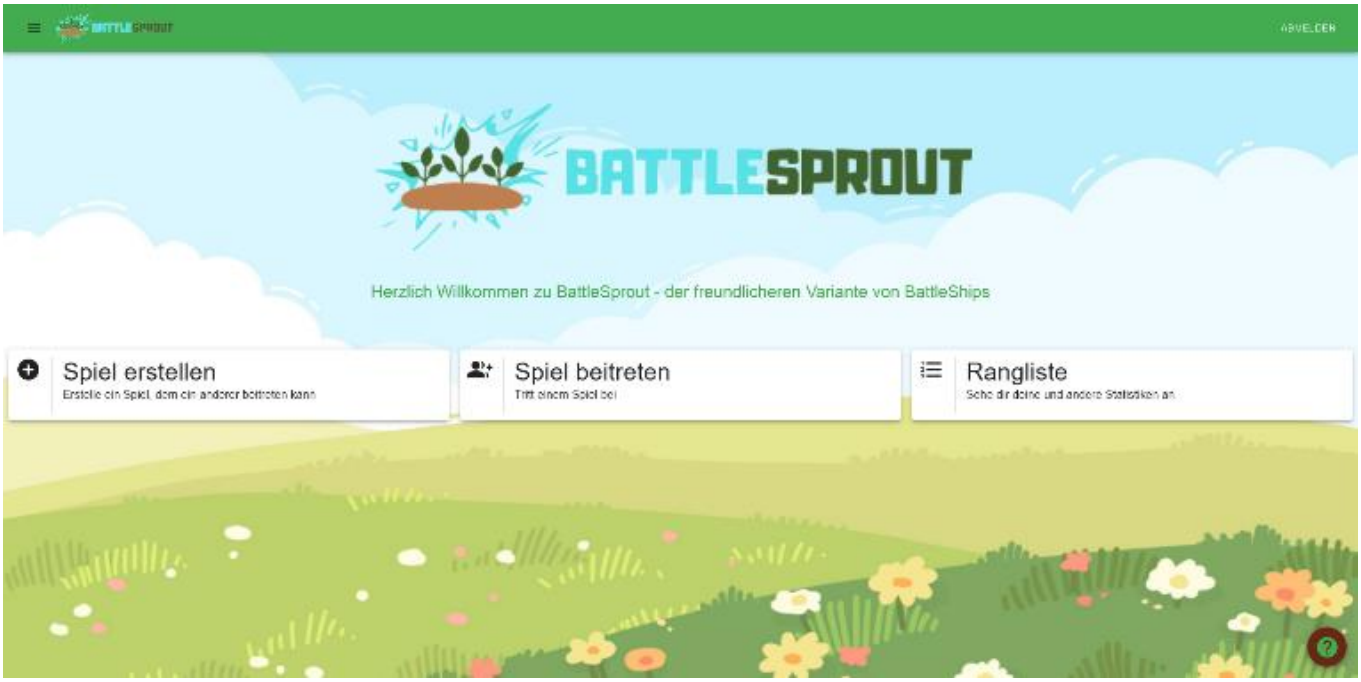
Eine weitere Möglichkeit besteht darin, die Anwendung auf andere Plattformen zu erweitern, z. B. mobile Geräte oder Tablets, um eine breitere Nutzerbasis anzusprechen. Dies erfordert möglicherweise Anpassungen an das Frontend, um das Platzieren der Pflanzen und das Werfen von Wasserbomben mit Touchscreen Eingabe zu ermöglichen.

Das Ausbauen des Nutzerprofils sowie der Freundes- und Rangliste kann in Erwägung gezogen werden, um dem Benutzer eine personalisierte Erfahrung zu bieten. Es soll beispielsweise in Zukunft möglich sein, seine Freunde über die Freundesliste zum Spiel einladen zu können.

Insgesamt bietet BattleSprout eine solide Grundlage für zukünftige Erweiterungen und Verbesserungen.

Durch die kontinuierliche Weiterentwicklung und das Hinzufügen neuer Funktionen kann die Anwendung attraktiver werden und die Benutzererfahrung der Anwendung verbessert werden.

Die genannten Vorschläge bieten einen Ausblick auf mögliche Entwicklungsrichtungen, die die Anwendung in Zukunft ansteuern könnte.


BATTLESPROUT
Herzlich Willkommen zu BattleSprout - der freundlicheren Variante von BattleShips
Spiel erstellen
Erstelle ein Spiel, dem ein anderer beitreten kann
Spiel beitreten
Tritt einem Spiel bei
Rangliste


Vorsicht - dein Nachbar wirft eine Wasserbombe...
Dein Beet
Beet von Unknown_User