# FireForceDefense
## Technical Report

Cameron Barbee, Tim Hoffmann, Christian Piffel, Tobias Schotter,
Sebastian Schuscha, Philipp Stangl, Thomas Stangl

## I. Einführung und Ziele

Im vorliegenden Projekt soll das Tower-Defense-Spiel „FireForceDefense“ erweitert werden. Ziel ist es, das Spiel um eine einfache Benutzerverwaltung, d.h. Registrierung und Anmeldung, zu ergänzen. Außerdem soll eine Rangliste geschaffen werden, wofür eine Spielstand-Speicherung notwendig ist.

## II. Bausteinsicht

Diese Sicht zeigt die statische Zerlegung des Systems in Bausteine sowie deren Beziehungen.

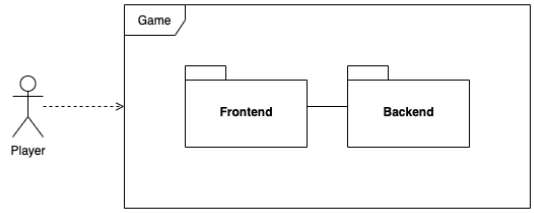


Fig. 1. Kontextabgrenzung

### A. Gesamtsystem

Die Anwendung basiert auf einer Client-Server-Architektur (1). Frontend und Backend kommunizieren über eine RESTful-API.

### B. Frontend

Dieser Abschnitt beschreibt die client-seitige Frontend-Architektur. Das Frontend wird unter Zuhilfenahme des Frameworks Vue.js realisiert.

Es ist selbst in mehrere Unter-Bausteine zerlegt:

*1) Model:* Dieser Baustein enthält die Logik für den Problembereich im Frontend. Die eigentlichen Anzeigekomponenten besitzen Referenzen auf die Instanzen der Klassen des Model-Bereichs, sodass Eingaben an das Model weitergegeben und dort verarbeitet werden können. Schließlich besitzt das Model Schnittstellen, mittels denen Informationen über den aktuellen Zustand abgefragt werden können, um basierend darauf die Anzeige anzupassen.

Im Model ist beispielsweise die Spiellogik und die Schnittstelle zum Speichern und Abrufen von Spielständen verortet.

*2) Components:* In diesem Modul sind die Vue-Komponenten gesammelt, mit denen die eigentliche Anzeige im Webbrowser realisiert wird. Die Komponenten sind dabei hierarchisch geordnet, so besteht ein Level beispielsweise aus einer Sidebar und der LevelMap, die LevelMap wiederum enthält einzelne Zellen und so weiter.

Die Komponenten behandeln alle Eingaben und senden bei Bedarf entsprechende Nachrichten an das Model.

*3) SCSS:* Hier werden zentral genutzte Stile im SCSS-Format abgelegt.

*4) Lang:* Dieser Baustein ist für die Internationalisierung, genauer gesagt die Übersetzung, zuständig. Aktuell ist lediglich eine deutsche Sprachvariante hinterlegt.

*5) Levels:* Dieses Modul enthält die Level-Definitionen. Eine Level-Definition beschreibt den initialen Aufbau des Spielfelds und die auftretenden Effekte. Jedes Level muss anhand der jeweiligen Definition im LevelManager des Model-Bereichs registriert werden.

*6) Cells, Contents und Effects:* In diesen drei Bausteinen sind die verschiedenen, konkreten Zell-, Inhalts- bzw. Effekt-Typen samt ihren jeweiligen Eigenschaften hinterlegt.

### C. Backend

Dieser Abschnitt beschreibt die server-seitige Backend-Architektur.

*1) Datenbank:* Die Speicherung der Daten erfolgt im dokumentenorientierten NoSQL-Datenbankmanagementsystem *MongoDB*. Zusätzlich wird die Bibliothek *mongoose* für das Object Data Modeling (ODM) verwendet.
Der folgende Teilbereich beschreibt die Unterteilung der Daten in drei Collections:

- *Accounts:* Hier werden die nötigen Informationen eines Benutzers gespeichert, um eine Login- und Registrierungsfunktion zu gewährleisten. Das ist einerseits ein einzigartiger Benutzername, eine E-Mailadresse, ein verschlüsseltes Passwort, sowie ein Datum, welches den Registrierungszeitpunkt festhält.
- *Refreshtokens:* In dieser Dokumentensammlung werden alle nötigen Informationen für das Session-Handling gespeichert. Das ist einerseits die von MongoDB automatisch erstellte ID der einzelnen Benutzer, um jedem Benutzer seinen entsprechenden Token zuweisen zu können. Dazu wird der entsprechende Token mit Verfallsdatum des Tokens, Erstellungsdatum des Dokumentes und die IP des Erstellers gespeichert.
- *Scores:* Diese Sammlung an Dokumenten speichert alle relevanten Aspekte des Spielstandes. Einerseits den

Benutzernamen, das Level, die erreichten Sterne, das übriggebliebene Geld, sowie die verwendete Zeit und die Anzahl an verbrannten Zellen. Diese Informationen bilden die Grundlage für die Erstellung der Rangliste.

*2) Laufzeitumgebung:* JavaScript-basierte Plattform Node.js mit dem serverseitigen Webframework ExpressJS.

## III. Verteilungssicht

Das Verteilungssicht beschreibt die Verteilung des Gesamtsystems, wichtige Begründungen für diese Verteilungsstruktur und die Zuordnung von Softwareartefakten zu Bestandteilen der Infrastruktur. Zentrale Bestandteile der Verteilungsstruktur sind (A) das Kubernetes Cluster, (B) das Datenbank-Cluster und (C) die GitLab CI/CD-Pipeline.

### A. Kubernetes Cluster

Für ein Kubernetes-Cluster wurde sich entschieden, aufgrund der Tatsache, das mehrere virtuelle Cluster (sog. Namespaces) auf demselben physischen Cluster unterstützt werden. Dadurch kann die CI/CD Pipeline im *Gitlab-managed-apps* Namespace laufen, währenddessen die Anwendung auf einem anderen Namespace bereitgestellt wird.

Das Kubernetes Cluster wird über den Elastic Kubernetes Service des Cloud-Providers Amazon Web Services bereitgestellt. Aus datenschutzrechtlichen Gründen werden nur Cloud Server, die der Verfügbarkeitszone Frankfurt (eu-central-1) angehörig sind, verwendet.

### B. Datenbank-Cluster

Das Datenbank-Cluster besteht aus drei Replikationen. Es stellt jeweils eine Datenbank für den Entwicklungs- und Produktionsbetrieb bereit. Näheres zur Datenbank wird im Abschnitt II-C.1 erklärt.

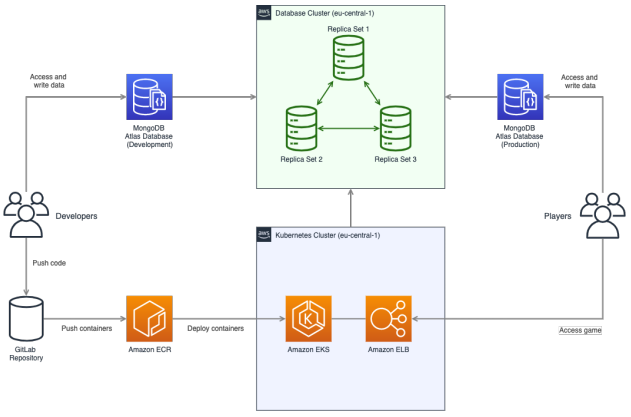


Fig. 2. Cloud Infrastruktur

### C. CI/CD-Pipeline

Für die Bereitstellung der CI/CD Pipeline ist ein Kubernetes-Executor[1] auf dem Kubernetes Cluster verantwortlich. Dieser verbindet sich mit der Kubernetes Cluster API und erstellt einen Pod für jeden GitLab CI Job. Dieser Pod besteht aus einem Build-Container und einem zusätzlichen Container für jeden Service. Die Pipeline umfasst das Erstellen, Testen und Bereitstellen der Anwendung. Folgende vier Stages durchläuft jeder Push in das Gitlab-Repository:

*1) Dependencies:* Installation der npm-Pakete in das Verzeichnis node_modules.

*2) Lint:* Durchführung statischer Code-Analysen.

*3) Build:* Alle notwendigen Bau-/Vorbereitungsaufgaben der Anwendung werden ausgeführt.

*4) Test:* Durchführung von Unit Tests, um die Korrektheit des Codes zu validieren.

Die letzten beiden Stages werden nur im main-Zweig des Repositories durchlaufen, um die Anwendung für den Produktionsbetrieb bereitzustellen.

*5) Dockerize:* Wenn alle Unit Tests bestanden sind, wird für Front- und Backend jeweils ein Docker-Image gebaut. Anschließend werden beide Images im Docker-Registry (Amazon ECR) abgelegt, damit sie für die nächste Stage (Deploy) verfügbar sind. Aufgrund fehlender Berechtigungen kann nicht von der privaten GitLab Container-Registry Gebrauch gemacht werden.

*6) Deploy:* In der letzten Phase werden die Docker Images aus der Docker-Registry auf dem Kubernetes Cluster als Container bereitgestellt. Die Anwendung ist dann, bis einschließlich dem Tag der Präsentation, öffentlich zugänglich.

## IV. Entwicklungsumgebung und Tools

### A. IDE

Das Projekt ist so ausgerichtet, dass es keine Beschränkung auf eine bestimmte IDE gibt. Die meisten Teammitglieder verwenden JetBrains Phpstorm.

### B. Paketverwaltung

Die Verwaltung der Abhängigkeiten erfolgt mit „npm“.

### C. Linting

In beiden Unterprojekten (Frontend und Backend) wird jeweils „eslint“ in Verbindung mit „prettier“ verwendet, um die Einhaltung der Codierrichtlinien zu gewährleisten.

Die Konfigurationen sind jeweils in den Dateien `eslintrc.js` und `prettierrc.js` hinterlegt.

### D. Build-Tools

*1) Backend:* Im Backend wird der Typescript Compiler „tsc“ verwendet, um die Dateien in ein Format zu überführen, welches mit `node` ausgeführt werden kann.

Die Konfiguration findet sich dabei in der Datei `tsconfig.json`.

*2) Frontend:* Im Frontend ist Webpack dafür zuständig, die Anwendung aus dem Quellcode zu erstellen. Dabei gibt es zwei Varianten: Für Entwicklungszwecke wird ein Webpack-Dev-Server (mit Reload-Funktionalität) zum Bereitstellen der Anwendung verwendet, während für den Produktiveinsatz nur die benötigten Zieldateien erstellt werden, die dann mit einer beliebigen Server-Software ausgeliefert werden können.

Die Webpack-Konfiguration erfolgt in den `webpack.config.*.js`-Dateien. Darin ist festgelegt,

dass die CSS-Bestandteile in der Ausgabedatei `main.css` gesammelt werden. Die Index-Datei und die Assets werden in das Ausgabeverzeichnis kopiert, die Vue-Single-File-Component-Dateien werden übersetzt und Typescript wird zu JavaScript kompiliert.

Ferner sind Alias-Namen für häufig genutzte Verzeichnisse definiert, um Pfadangaben zu vereinfachen.

### E. Unit Tests

Unit-Tests werden mit „Jest“ realisiert. Snapshot-Tests für die Frontend-Komponenten sind dabei durch die Pakete „vue-jest“ und „vue-test-utils“ möglich.

## V. Detailinformationen

Im Folgenden werden die beiden Bereiche „Session Handling“ und „Rangliste“, welche der Hauptfokus der diesjährigen Arbeiten waren, etwas genauer betrachtet.

### A. Session Handling

Das Session Handling basiert auf dem Konzept der „Sliding-Sessions“. Für die Implementierung des Konzepts werden „JSON-Web-Tokens“[2] verwendet.

*1) Token-Arten:* Man unterscheidet zwischen folgenden zwei Token-Arten:

- *Access Token:*
  Enthalten alle notwendigen Informationen für den Zugriff auf eine Ressource. Access Token sind kurzlebig und nur für 1 Stunde gültig.
- *Refresh Token:*
  Enthalten Informationen, um einen neuen Access Token zu erhalten, nachdem der Alte abgelaufen ist, oder der erstmalige Zugriff auf eine Ressource vom Server angefordert wird. Refresh Token sind langlebig und für 7 Tage gültig.

Jede Sitzung (engl. Session) läuft nach einer gewissen Zeit der Inaktivität ab. Verwendet der Benutzer einen abgelaufenen Access Token, wird die Sitzung als inaktiv betrachtet und ein neuer Access Token ist erforderlich. Dieser kann mit einem Refresh Token, der auf dem Client als „HTTP-Only Cookie“ hinterlegt ist, bezogen werden (Cookie-based Session Management). Näheres zur Speicherung des Refresh Token befindet sich im Abschnitt VI-B.

*2) Sitzungskennung:* Eine Sitzungskennung wird einem Refresh Token zugeordnet. Die mit einer Sitzungskennung verbundenen Daten befinden sich in der Datenbank (siehe II-C.1). Eine Sitzungskennung ist:

- zufällig generiert durch einen „Pseudorandom number generator“ (PNRG).
- im Cookie gespeichert.

### B. Rangliste

Die Spielergebnisse werden nach der Beendigung eines jeden Levels als Dokument in der „scores“-Collection der MongoDB abgelegt.

Basierend darauf wird backendseitig eine GET-Schnittstelle bereitgestellt, über welche – unter optionaler Angabe eines Levels, nach dem gefiltert werden soll – die aggregierten Spielstandsdaten abgefragt werden können. Diese Aggregation führt dazu, dass es pro Benutzername einen Eintrag gibt, welcher die aufaddierte Gesamtzahl der erreichten Sterne, die durchschnittlich benötigte Spieldauer, das durchschnittlich angesparte In-Game-Geld-Guthaben, sowie die durchschnittliche Anzahl der abgebrannten Felder enthält.

Auch die Sortierung erfolgt bereits im Backend, und zwar nach folgenden Kriterien (in absteigender Priorität geordnet): Sternenanzahl (absteigend), Anzahl der verbrannten Felder (aufsteigend), Geld-Guthaben (absteigend) und Spielzeit (absteigend).

Im Frontend werden diese Daten durch eine HTTP-Anfrage beschafft und durch eine Vue-Komponente auf einer eigenen Seite angezeigt. Dabei haben Benutzer*innen die Möglichkeit, entweder eine Gesamtansicht über alle Level hinweg oder eine nach einem bestimmten Level gefilterte Ansicht auszuwählen.

Die eigene Ranglisten-Position wird hervorgehoben und fixiert angezeigt.

## VI. Sicherheitskonzepte

Dieser Abschnitt beschreibt die angewendeten Sicherheitskonzepte, die für eine sichere Produktionsumgebung notwendig sind.

### A. Passwort Hashing

Bei der Erstellung eines neuen Benutzerkontos wird vor der Speicherung in der Datenbank ein „Hash“ für das gesetzte Passwort berechnet. Vor dem Hashing wird eine zufällige Zeichenfolge, ein sogenannter „Salt“, dem Passwort hinzugefügt. Dadurch werden Kennwörter nicht jedes Mal auf dieselbe Weise gehasht. Somit können Angriffe wie Nachschlagetabellen, Wörterbuchangriffe oder Brute-Force unterbunden werden.

Wenn ein Benutzer versucht sich anzumelden, wird der Hash des eingegebenen Kennworts mit dem in der Datenbank gespeicherten Hash verglichen. Stimmen die Hashes überein, kann der Benutzer auf das Konto zugreifen.

### B. Refresh Token Speicherung

Refresh Token sind langlebig. Das bedeutet, wenn ein Client einen Refresh Token vom Server erhält, muss dieser Token sicher gespeichert werden, damit dieser nicht von potenziellen Angreifern entwendet werden kann. Wenn ein Refresh Token entwendet wird, kann er verwendet werden, um neue Access Token zu erhalten und auf Ressourcen zugreifen, bis der Refresh Token abläuft. Deshalb werden Refresh Token als Cookie mit dem HttpOnly-Attribut gespeichert. Dies verhindert, dass mittels einem „Cross-Site-Scripting (XSS)“-Angriff die Sitzungskennung gestohlen werden kann.

## References


[1] GitLab. Gitlab Runner: Kubernetes executor. `https://docs.gitlab.com/runner/executors/kubernetes.html`

[2] JWT. JSON-Web-Token: RFC 7519. `https://datatracker.ietf.org/doc/html/rfc7519`