

Realisierungsbericht

Status	Abgeschlossen
Projektname	xServer
Projektleiter	Maurice Däppen
Auftraggeber	VALVE
Autoren	Maurice Däppen, Patrick Aeschlimann, Lenny Herren, Mika Hannappel
Verteiler	Maurice Däppen, Patrick Aeschlimann, Lenny Herren, Mika Hannappel, Christian Kissling

Änderungskontrolle, Prüfung, Genehmigung

Version	Datum	Beschreibung, Bemerkung	Name oder Rolle
V1.0.0	22.05.2024	Erstellung	Maurice Däppen

Definitionen und Abkürzungen

Begriff / Abkürzung	Bedeutung

Referenzen

Referenz	Titel, Quelle
[1]	
[2]	
[3]	

Inhaltsverzeichnis

1	Zusammenfassung	4
2	Technische Detailspezifikation	4
2.1	Systemdesign	5
2.1.1	Struktur des Systemdesigns	5
2.1.2	Beschreibung der Elemente	7
2.1.2.1	Frontend	7
2.1.2.1.1	Framework	7
2.1.2.1.2	Libraries	7
2.1.2.1.3	Struktur	9
2.1.2.2	Backend	9
2.1.2.2.1	Framework	9
2.1.2.2.2	Libraries	9
2.1.2.2.3	Struktur	11
2.1.3	Dynamik	11
2.1.3.1	Sign Up	12
2.1.3.2	Sign in	13
2.1.3.3	Server erstellen	14
2.2	Schnittstellendefinitionen	15
2.2.1	Frontend → Backend	15
2.2.2	DB	16
2.2.3	Cluster	17
2.2.3.1	Konfiguration	17
2.2.3.2	Deployment	17
2.2.3.3	Service	18
2.3	Datenmodel	20
2.4	Sicherheit	20
2.4.1	DB	20
2.4.2	Kommunikation Frontend ↔ Backend	20
2.4.3	JWT	20
2.4.4	Disclaimer	20
3	Systemdokumentation	21
3.1	Konfigurations-Dokumentation	21
3.1.1	Development-Konfiguration	21
3.1.2	Lokale Konfiguration	21
3.1.3	Prod Konfiguration	21
3.2	Benutzerhandbuch	22
3.2.1	Systemübersicht	22
3.2.2	Anwenderfunktionalität	23
3.3	Integrations- und Installationshandbuch	24
3.4	Supporthandbuch	24
3.4.1	Massnahmen bei Benutzerproblemen	24
3.4.2	Massnahmen bei technischen Problemen	24
3.4.3	Anhang zum Supporthandbuch	24
4	Test	25
4.1	Unit Test	25
4.2	Systemtest - Testspezifikation	25
4.2.1	Kritikalität der Funktionseinheit	25
4.2.2	Testanforderungen	26
4.2.3	Testverfahren	26
4.2.4	Testkriterien	28
4.2.4.1	Checklisten	28
4.2.4.2	Endkriterien	28
4.2.5	Testfälle	29
4.3	Testprozedur	29
4.3.1	Test 1	29
4.3.2	Test 2	30
4.3.3	Test 3	30
4.3.4	Test 4	31
4.3.5	Test 5	31
4.3.6	Test 6	32

4.4	Testprotokoll	32
5	Antrag auf Freigabe der nächsten Projektphase	34

Abbildungsverzeichnis

1 Zusammenfassung

Dieser Realisierungsbericht bietet eine umfassende Übersicht über die Umsetzung von xServer, einer Plattform zur Verwaltung und Bereitstellung von Gameservern. Er dokumentiert die Schritte und Massnahmen während der Realisierungsphase, einschliesslich der Implementierung der Systemanforderungen, Integration technischer Schnittstellen, Entwicklung der Benutzeroberfläche und Durchführung von Tests.

Das Dokument beschreibt die funktionalen Anforderungen sowie die Massnahmen zur Sicherstellung der Informationssicherheit und des Datenschutzes. Es enthält eine Übersicht der Systemarchitektur, der internen und externen Schnittstellen sowie der Benutzeroberfläche. Darüber hinaus werden die durchgeführten Implementierungsschritte und erreichten Meilensteine dargestellt.

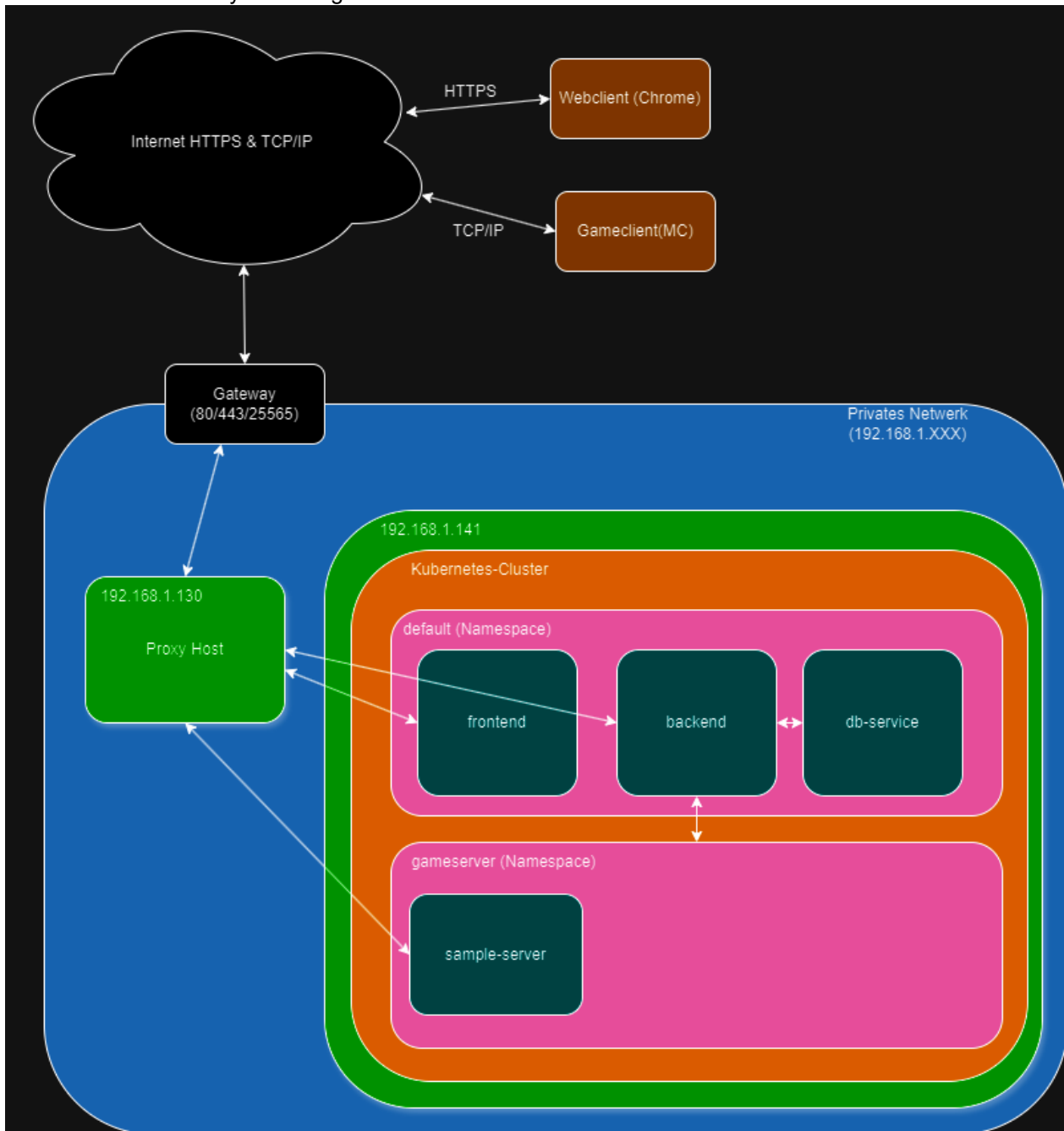
Ein Schwerpunkt liegt auf dem Testkonzept, der Teststrategie und den Testfällen, ergänzt durch eine Zusammenfassung der Testergebnisse und durchgeführter Fehlerbehebungen. Abschliessend fasst der Bericht die Erkenntnisse aus der Realisierungsphase zusammen und gibt Empfehlungen für die nächsten Schritte und zukünftige Optimierungen.

2 Technische Detailspezifikation

In diesem Abschnitt werden die technischen Detailspezifikationen von xServer beschrieben. Die Systemarchitektur, wie im Konzeptbericht dargestellt, wird während der Realisierungsphase konkret umgesetzt und bei Bedarf verfeinert. Dies umfasst die genaue Konfiguration der einzelnen Systemelemente, die Anpassung bestehender Komponenten und die Integration neuer Elemente.

2.1 Systemdesign

2.1.1 Struktur des Systemdesigns



Wir konnten uns eigentlich ziemlich gut an die bereits geplante System Architektur halten, jedoch gab es einige Optimierungen. Insgesamt konnten wir eine funktionierende und einigermaßen komplexe Architektur umsetzen. Diese Umsetzung war auch der grösste Teil unserer Arbeit da wir viel nach Lösungen und Möglichkeiten suchen mussten und diese dann auch umsetzen mussten. Schlussendlich haben wir uns für obenstehende Architektur entschieden.

Das private Netzwerk

Wir haben ein privates Netzwerk aufgesetzt, in welchem wir alle unsere Komponenten hosten. Dieses Netzwerk verfügt über ein Gateway mit einer Firewall welches nur die Ports 80, 443 und 25565 offenhält. Dazu haben wir uns aus Sicherheitsgründen entschieden. 80 und 443 sind geöffnet für unsere Website sowie das Backend. 25565 ist der Standardport für Minecraftserver und deshalb haben wir diesen auch noch geöffnet. Evtl. müssen wir in der Zukunft noch weitere Standardports von anderen Games öffnen. Wir haben einen DNS Eintrag erstellt welcher für alle Subdomains von unserer Domain (*.xserver.space) auf dieses

Netzwerk zeigt. Somit kommen alle Anfragen auf unser Netzwerk wie z.B. `host.xserver.space` für unsere Website oder auch `uuid.xserver.space` für irgend einen Gameserver.

In diesem Netzwerk befinden sich 2 Hosts. Zum einen einen Host für unseren Proxy Manager und zum anderen einen Host mit unserem Kubernetes Cluster.

Proxy Manager

Unser Proxymanager nutzt Traefik die Subdomains aller Anfragen, welche auf `*.xserver.space` kommen, aufzulösen und diese entsprechen weiterzuleiten. Traefik hat die Vorteile, dass es nicht nur HTTP/S unterstützt sondern auch andere Protokolle wie TCP/IP welches von Minecraft-Clients verwendet wird. Zudem verfügt Traefik über eine API, mit welcher man während der Laufzeit Einträge erstellen, bearbeiten und löschen kann. Zudem kann man mithilfe von Let's Encrypt direkt SSL-Zertifikate für die entsprechenden Domains erstellen und automatisch erneuern. So haben wir auch die SSL-Zertifikate von `host.xserver.space` und `backend.xserver.space` erstellt.

Kubernetes Cluster

Für unseren Kubernetes Cluster nutzen wir `microk8s`. Dieser Cluster läuft auf einem Host und hat Ressourcen von 128 GB RAM, 4 TB SSD und 64 CPU-Cores verfügt. Diese stehen für das Hosting von Gameserver zu Verfügung, jedoch lassen sich diese Ressourcen dynamisch erweitern. Im Cluster selbst befinden sich 2 namespaces. Einmal der default Namespace und der gameserver Namespace. Im default Namespace laufen unsere eigenen Applikationen, also das Frontend, das Backend und die DB. Im gameserver Namespace laufen die Gameserver. Dies ist auch wieder aus Sicherheitsgründen getrennt worden damit jeder Server unabhängig von den anderen laufen kann und vor allem, dass unsere DB möglichst isoliert ist.

Detailkomponenten

Im folgenden Abschnitt gehen wir genauer auf die Komponenten ein, welche unsere Applikation ausmachen und welche wir selbst entwickelt haben. Damit diese Komponenten in unserem Cluster laufen benötigen wir Images. Wir haben uns für Dockerimages entschieden. Die Dockerimages sind in unserem öffentlichen Containerregistry und werden von dort gepulled. Diese Images werden jedes Mal gebuildet und ins Containerregistry gepusht, wenn wir ein Tag auf master erstellen oder eine PR auf master mergen. Somit haben wir immer Zugriff auf die neusten Images. Diese werden leider nicht automatisch in unserem Cluster installiert und wir müssen die noch manuell machen (Siehe Konfigurations-Dokumentation). Front- und Backend nutzen Nodeport und die DB nutzt ClusterIP damit sie auch schön isoliert ist.

Frontend

Unser Frontend wurde mit Angular umgesetzt und dient als Plattform um die Server zu erstellen, verwalten und löschen. Um das Frontend im Browser zu verwenden ist es nötig JavaScript zu aktivieren. Das Docker Image nutzt ein Node Image um die App zu builden und ein nginx Image um die App zu runnen. Um das Frontend im Kubernetes Cluster laufen zu lassen benötigen wir ein Deployment sowie einen Service. Diese Ressourcen werden mithilfe von Kubernetes-Configs erstellt. Mithilfe von NodePort lässt sich via Port 30000 auf unser Frontend zugreifen. Dieser Port wird auch von unserem Proxy angesteuert.

Backend

Unser Backend wurde mit Express JS umgesetzt und dient als Schnittstelle zwischen dem Frontend und der DB. Zudem validiert und handhabt dieses auch Userdaten handelt den Zugang. Auch hier haben wir uns für ein Dockerimage entschieden um das Backend in unserem Cluster laufen zu lassen. Für das Dockerimage wurde hier nur ein Node-Image verwendet da man ein Express JS Backend nicht zuerst builden muss. Neben dem Deployment und dem Service für unseren Cluster benötigt das Backend noch eine Custom Role um andere Kubernetes Ressourcen (Unsere Gameserver) zu managen. Wichtig ist dabei, dass die Berechtigungen nur für den gameserver Namespace gelten. Mithilfe von NodePort lässt sich via Port 30001 auf unser Backend zugreifen. Dieser Port wird auch von unserem Proxy angesteuert.

DB-Service

Für unsere DB nutzen wir eine MongoDB da wir diese bereits kennen und leicht zu handhaben ist. Hier konnten wir direkt das offizielle Mongo Image nutzen und in unsere App einbinden. Da unsere DB isoliert laufen soll nutzen wir hier in unserem Cluster ClusterIP somit können nur App im selben Namespace (Front- & Backend) auf die DB zugreifen. Natürlich könnte man manuell mit `kubectf` ein Portforwarding machen jedoch ist so etwas nur temporär. Neben einem Deployment und einem Service benötigt unsere DB noch einen PersistentVolume um die Daten in einem eigenen Volume zu speichern. Somit verlieren wir unsere Daten nicht wenn die DB abschmiert.

2.1.2 Beschreibung der Elemente

In diesem Unterkapitel gehen wir nun noch etwas genauer auf die einzelnen Hauptkomponenten unserer App ein.

2.1.2.1 Frontend

2.1.2.1.1 Framework

Wir haben unser Frontend mit Angular 17.2.0 umgesetzt. Angular ist ein von Google entwickeltes Framework für die Erstellung von Single-Page-Webanwendungen. Es basiert auf TypeScript und bietet Entwicklern eine robuste Struktur und umfangreiche Bibliotheken zur Entwicklung dynamischer und reaktiver Webapplikationen. Angular ermöglicht die Trennung von Logik und Darstellung durch Komponenten, verwendet eine deklarative Programmierung mit Templates und unterstützt Two-Way-Databinding, um eine nahtlose Synchronisation zwischen Model und View zu gewährleisten. Dank einer Vielzahl von eingebauten Werkzeugen und einer aktiven Community erleichtert Angular die Entwicklung, das Testen und die Wartung komplexer Anwendungen.

2.1.2.1.2 Libraries

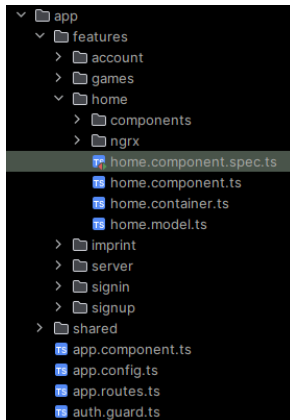
Da wir mit Angular allein nicht alles machen konnten, was wir vorhatten, haben wir uns das Leben mit einigen Libraries erleichtert. Bei dem Einbinden der Libraries haben wir darauf geachtet, dass alle aktiv gewartet werden und somit ein geringeres Risiko für sicherheitsrelevante Probleme besteht.

Name (Version)	Beschreibung
Angular Material (v17.2.1)	Damit wir uns mehr auf das Entwickeln konzentrieren können haben wir uns mit Angular Material eine Bootstrap Library für Angular hinzugezogen. Angular Material ist eine UI-Komponentenbibliothek, die speziell für Angular-Anwendungen entwickelt wurde und auf den Material Design-Richtlinien von Google basiert. Sie bietet eine Vielzahl vorgefertigter, stilvoller und anpassbarer Komponenten wie Schaltflächen, Formularelemente, Dialoge, Navigationselemente und mehr, die uns helfen, konsistente und benutzerfreundliche Benutzeroberflächen zu erstellen.
jwt-decode (v4.0.0)	Da wir unsere Authentifizierungssystem selbst umgesetzt haben brauchen wir im Frontend eine Library um die Werte in den Tokens wie z.B. das Ablaufdatum auszulesen. Da kommt jwt-decode ins Spiel. jwt-decode ist ein JavaScript-Tool, das zur Dekodierung von JSON Web Tokens (JWT) verwendet wird. Es ermöglicht uns das Auslesen des Inhalts von JWTs ohne die Notwendigkeit eines geheimen Schlüssels oder einer Validierung. Diese Bibliothek wird häufig in Client-Anwendungen eingesetzt, um die im Token enthaltenen Daten, wie z.B. Benutzerinformationen und Berechtigungen, schnell und einfach zu extrahieren.
RxJS (v7.8.0)	Wir benutzen RxJS nur indirekt in unserem Frontend. Denn RxJS (Reactive Extensions for JavaScript) ist ein leistungsfähiges Tool für reaktive Programmierung in JavaScript, das mit Observables arbeitet, um asynchrone Datenströme zu handhaben. In Kombination mit NgRx, einer Library für State-Management in Angular-Anwendungen welche später noch behandelt wird, wird RxJS essenziell, da NgRx auf Observables und Operatoren von RxJS basiert. Dadurch können wir komplexe Zustandsänderungen und

	Nebenwirkungen in Angular-Applikationen deklarativ und reaktiv steuern, was zu einer verbesserten Wartbarkeit und Testbarkeit des Codes führt.
NgRx (v17.1.0)	NgRx ist ein Redux-ähnliches State-Management-Tool für Angular-Anwendungen. Es ermöglicht die Verwaltung des Anwendungszustands in einem zentralen Store und stellt sicher, dass der Zustand vorhersagbar und leicht zu debuggen ist. NgRx verwendet Actions, Reducers und Selectors, um Änderungen am Zustand zu verwalten und Daten an Komponenten zu liefern. Dies fördert eine klare Trennung der Zustandslogik von der Darstellung und erleichtert die Wartung und Skalierung von Angular-Projekten. In Kombination mit Effects können asynchrone Tasks wie REST-Abfragen ganz einfach reaktiv umgesetzt werden und es muss sich nicht wirklich um die Promises gekümmert werden.
Jest (v29.7.0)	Für das Testing nutzen wir Jest als grundlegendes Testframework. Es ist ähnlich wie JUnit in Java und lässt sich auch sehr ähnlich verwenden. Jest ist ein leistungsstarkes und benutzerfreundliches JavaScript-Framework für das Testen von Anwendungen. Entwickelt von Facebook, ist Jest besonders geeignet für das Testen von React-Anwendungen, unterstützt aber auch andere JavaScript-Bibliotheken und -Frameworks wie z.B. Angular was wir nutzen. Jest bietet Funktionen wie Snapshots, um UI-Komponenten zu testen, integrierte Mocking-Methoden, welche wir jedoch nicht verwenden, um Abhängigkeiten zu isolieren, und eine umfassende Testabdeckung.
ts-mockito	Wie bereits bei Jest gesagt nutzen wir die mocking Funktionen von Jest nicht da diese sich als etwas problematisch beim Testen von NgRx Komponenten herausstellen. Da kommt ts-mockito in Spiel. Es lässt sich mit Mockito aus Java vergleichen und verwendet sich fast gleich. ts-Mockito ist eine Library für TypeScript, die das Erstellen und Verwalten von Mock-Objekten vereinfacht. Sie ermöglicht uns, Tests zu schreiben, indem sie Objekte simulieren und deren Verhalten kontrollieren, um verschiedene Szenarien zu testen. Die Bibliothek unterstützt typische Mocking-Funktionen wie das Festlegen von Rückgabewerten, das Überprüfen von Aufrufzeiten und -reihenfolgen sowie das Simulieren von Ausnahmen. Durch eine intuitive API und Integration in Testframeworks wie Jasmine oder Jest(Was wir nutzen) hilft ts-Mockito dabei, saubere und wartbare Unit-Tests zu erstellen.

2.1.2.1.3 Struktur

Unser Frontend ist in fachliche Packages aufgeteilt:



Jedes fachliche Package beinhaltet ein model-File in welchem unsere DTO-Objekte sich befinden. Jede fachliche Einteilung ist gleichzeitig auch eine Page in unserer App. So ist zum Beispiel das Server Package die Seite von «/server». Wie im Screenshot zu sehen ist verfolgen wir eine Container-Component Architektur. Dabei ist der Container die äusserste Komponente und alles andere befindet sich innerhalb des Containers. Der Container ist für die Interaktionen mit den NgRx-Komponenten zuständig, dass heisst er löst Actions aus und subscribed zu Werten im Store. Der Component hingegen ist rein für die Darstellung und das Handling von Userinput zuständig. In der App selbst gibt es einen globalen Store und jedes fachliche Package verfügt über eigene NgRx Komponenten wie Actions, Effects, Services, Reducers und Selectors. Zudem gibt es noch einen Ordner, in welchem die Kubernetes Ressourcen liegen. Dort sind die Files um das Frontend selbst zu deployen.

Dokumentiert wird der Code mit inline Comments da wir für JavaScript keine gute Option gefunden haben, wie man den Code besser kommentieren kann. Wir werden dies noch in einer Demo zeigen.

2.1.2.2 Backend

2.1.2.2.1 Framework

Für unser Backend nutzen wir Express JS. Express JS ist ein minimalistisches und flexibles Webanwendungs-Framework für Node.js, das eine robuste Reihe von Funktionen für Web- und Mobile-Anwendungen bietet. Es erleichtert die Entwicklung von Webservern und APIs, indem es eine einfache und klare Struktur bereitstellt. Mit Express.js können wir schnell Routen definieren, Middleware einfügen und Anfragen sowie Antworten effizient verwalten. Dank seiner leichten und unaufdringlichen Natur lässt sich Express.js leicht an verschiedene Projektanforderungen anpassen, wodurch es zu einem der am häufigsten verwendeten Frameworks in der Node.js-Entwicklung wird.

2.1.2.2.2 Libraries

Da Express JS ein Node.js Framework ist lässt es sich ganz einfach mit vielen anderen Node.js Libraries kombinieren und man kann für fast jede Tätigkeit eine Library finden welche uns das Leben einfacher macht. Auch im Backend haben wir beim Einbinden der Libraries darauf geachtet, dass alle aktiv gewartet werden und somit ein geringeres Risiko für sicherheitsrelevante Probleme besteht.

Name (Version)	Beschreibung
kubernetes/client-node (v0.21.0)	Für das Erstellen, Verwalten und Löschen unserer Gameserver nutzen wir kubernetes/client-node. Kubernetes/client-node ist eine offizielle JavaScript/TypeScript-Client-Library für die Kubernetes API. Sie ermöglicht es uns, Kubernetes-Ressourcen und -Dienste aus unserem Backend heraus zu verwalten. Mit dieser Library können wir Kubernetes-Cluster programmatisch steuern, Ressourcen wie Pods, Dienste und Deployments erstellen, aktualisieren und löschen sowie Informationen zu deren Status abrufen. Die client-node-Library abstrahiert die Komplexität der direkten Interaktion mit der Kubernetes API und erleichtert die Integration von Kubernetes-Funktionalitäten in Node.js-basierte Anwendungen.

	Wichtig ist dabei dass unsere Backend eine spezielle Rolle im Cluster benötigt um diese Aktionen auszuführen.
bcrypt (v5.1.1)	Für das hashen und überprüfen unserer Passwörter benutzen wir bcrypt. bcrypt ist ein Werkzeug für sicheres Hashing von Passwörtern. Es verwendet den Blowfish-Verschlüsselungsalgorithmus und bietet eine adaptive Hash-Funktion, die durch die Festlegung einer sogenannten "Cost-Factor" die Komplexität der Berechnung steuert. Dies macht es für Angreifer schwieriger, Passwörter mittels Brute-Force-Attacken zu knacken, da die Berechnungen zeitaufwendiger werden. bcrypt ist besonders nützlich für die Speicherung von Passwörtern in Datenbanken, da es zusätzliche Sicherheitsmechanismen wie Salting (das Hinzufügen eines zufälligen Werts zu jedem Passwort) integriert, um die Sicherheit weiter zu erhöhen.
dotenv (v16.4.5)	Da wir in den verschiedenen Umgebungen verschiedene Werte für Variablen benötigen haben wir dotenv in unser Projekt hinzugefügt. Dotenv ist eine Library, die verwendet wird, um Umgebungsvariablen aus einer Datei namens .env zu laden und diese in einer Anwendung verfügbar zu machen. Diese Datei enthält Schlüssel-Wert-Paare, die sensible Informationen wie API-Schlüssel, Datenbankzugangsdaten oder Konfigurationsparameter speichern. Durch die Verwendung von Dotenv können wir unsere Konfigurationsdaten getrennt vom Quellcode halten, was die Sicherheit und Flexibilität erhöht. Die Bibliothek liest die .env-Datei beim Start der Anwendung ein und stellt die darin enthaltenen Variablen dem Prozess zur Verfügung, sodass sie über process.env (in Node.js) oder ähnliche Mechanismen in anderen Programmiersprachen abgerufen werden können.
jsonwebtoken (v9.0.2)	Da wir unser Login etc. selbst umgesetzt haben brauchen wir natürlich ein Tool um unsere Tokens zu erstellen und validieren. jsonwebtoken ist ein beliebtes Node.js-Modul, das JSON Web Tokens (JWTs) erstellt, verifiziert und dekodiert. JWTs sind kompakte, URL-sichere Tokens, die häufig für die Authentifizierung und Autorisierung in Webanwendungen verwendet werden. Die Bibliothek ermöglicht das Signieren von Tokens mit verschiedenen Algorithmen (wie HMAC oder RSA), um die Integrität und Authentizität der übertragenen Daten zu gewährleisten. Mit jsonwebtoken können wir ganz einfach sicherstellen, dass Benutzer identifiziert und autorisiert sind, indem wir Tokens generieren, die Server und Client auf einfache Weise austauschen und validieren können.
mongodb (v6.5.0)	Da unser Backend als Schnittstelle zwischen Frontend und DB dient benötigen wir natürlich auch ein Tool um sicher und einfach auf unsere DB zuzugreifen. Die MongoDB-Library ermöglicht die Interaktion mit einer MongoDB-Datenbank. Sie bietet eine Programmierschnittstelle, um CRUD-Operationen (Erstellen, Lesen, Aktualisieren,

	Löschen) durchzuführen und komplexe Abfragen zu erstellen. Wir können damit Dokumente in JSON-ähnlichen BSON-Formaten speichern und abfragen. Die Bibliothek unterstützt Verbindungen zu MongoDB-Servern, Transaktionen und bietet Werkzeuge für die Indizierung und Aggregation von Daten. Sie ist in verschiedenen Programmiersprachen verfügbar, darunter Python, JavaScript und Java, und erleichtert die Integration von MongoDB in Anwendungen.
uuid (v9.0.1)	Da wir für unsere id's in für das Speichern in unserer DB selbst verwalten und kontrollieren möchten verwenden wir uuid. Die uuid-Library in JavaScript ist ein populäres Werkzeug zum Generieren von Universally Unique Identifiers (UUIDs). Diese werden häufig für eindeutige Kennzeichnungen verwendet, wie zum Beispiel für Datenbankeinträge oder als eindeutige Bezeichner in Webanwendungen. Die Bibliothek unterstützt verschiedene UUID-Versionen, darunter die weit verbreiteten Versionen 1 und 4. UUIDv1 basiert auf Zeit und Hardware-Adresse (MAC-Adresse), während UUIDv4 zufallsbasiert ist.

2.1.2.2.3 Struktur

Auch im Backend setzen wir auf eine Struktur von fachlichen Packages. Zudem gibt es ein Main-File namens server.js welches alle Endpoints registriert und bereitstellt. Zudem gibt es ein geteiltes Util-File welche Middleware beinhaltet für die Token Validierung. Zudem gibt es noch einen Ordner, in welchem die Kubernetes Ressourcen liegen. Zum einen sind dort die Files um das Backend selbst zu deployen und zum anderen auch noch Configs mit welchen man Gameserver deployen kann.

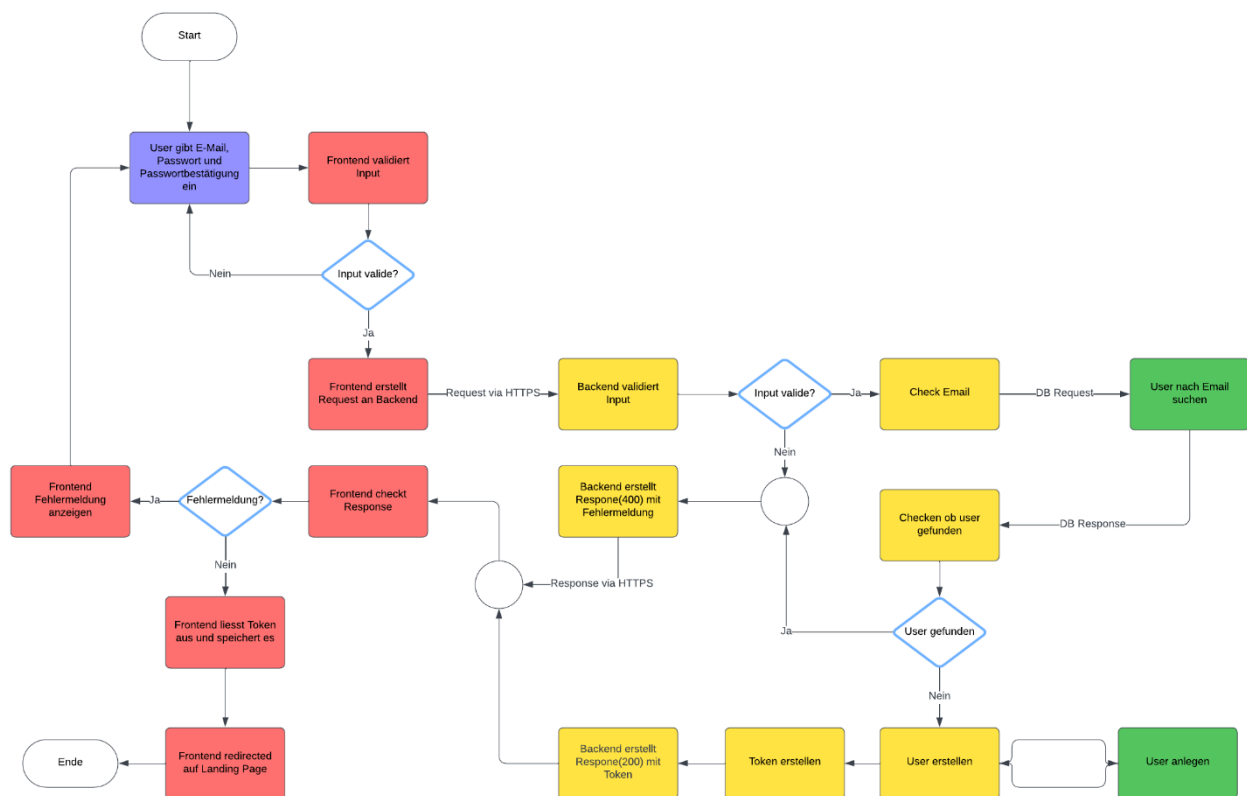
Dokumentiert wird der Code mit inline Comments da wir für JavaScript keine gute Option gefunden haben, wie man den Code besser kommentieren kann. Wir werden dies noch in einer Demo zeigen.

2.1.3 Dynamik

Da unser System noch nicht allzu viele Features und Prozesse hat wollen wir auf die wichtigsten eingehen. Da wir nicht direkt objektorientiert vorgehen benötigen wir ein Diagramm, welches sowohl Systemübergreifende Prozesse darstellen kann, jedoch auch in einer nicht objektorientierten Umgebung funktionieren. Wir dachten an ein Sequenzdiagramm und dachten, dass dies eher unpassend ist, da es viele Verzweigungen gibt sowie ein Sequenzdiagramm in einer rein objektorientierten Umgebung besser geeignet ist. Deshalb stellen wir nun unsere 3 Kernprozesse in Flowchart dar.

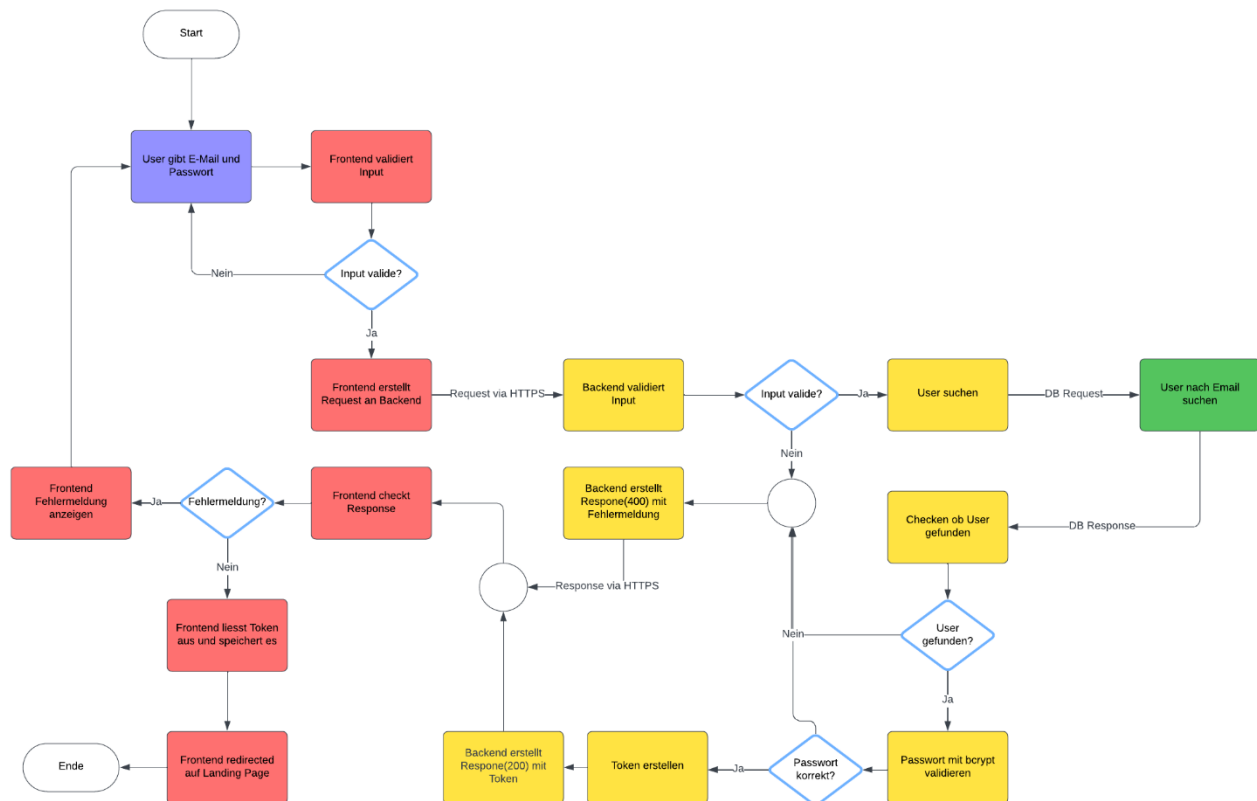
2.1.3.1 Sign Up

Beim Sign Up geht es darum einen Account zu erstellen. Das folgende Diagramm zeigt den Prozess der Erstellung von einem Account. Das Ganze beginnt, indem der User im Frontend seine Daten eingibt. Diese Daten werden vom Frontend validiert, falls diese nicht in Ordnung sind, beginnt der Prozess von vorne. Falls die Daten in Ordnung sind, wird eine Backend Request getätigt und das Backend validiert die Daten n nochmals mithilfe der DB. Falls die Daten nun nicht in Ordnung sind, gibt es erneut eine Fehlermeldung und diese wird ans Frontend geschickt. Falls die Daten in Ordnung sind, wird eine Usererstellung, das dazugehörige Token wird erstellt und ans Frontend gespeichert. Falls das Frontend eine Fehlermeldung von Backend erhält, zeigt das Frontend diese an und der Prozess beginnt von vorne. Falls das Frontend keine Fehlermeldung erhält, wird das Token gespeichert, der User wird auf die Landingpage weitergeleitet und der Prozess ist zu Ende.



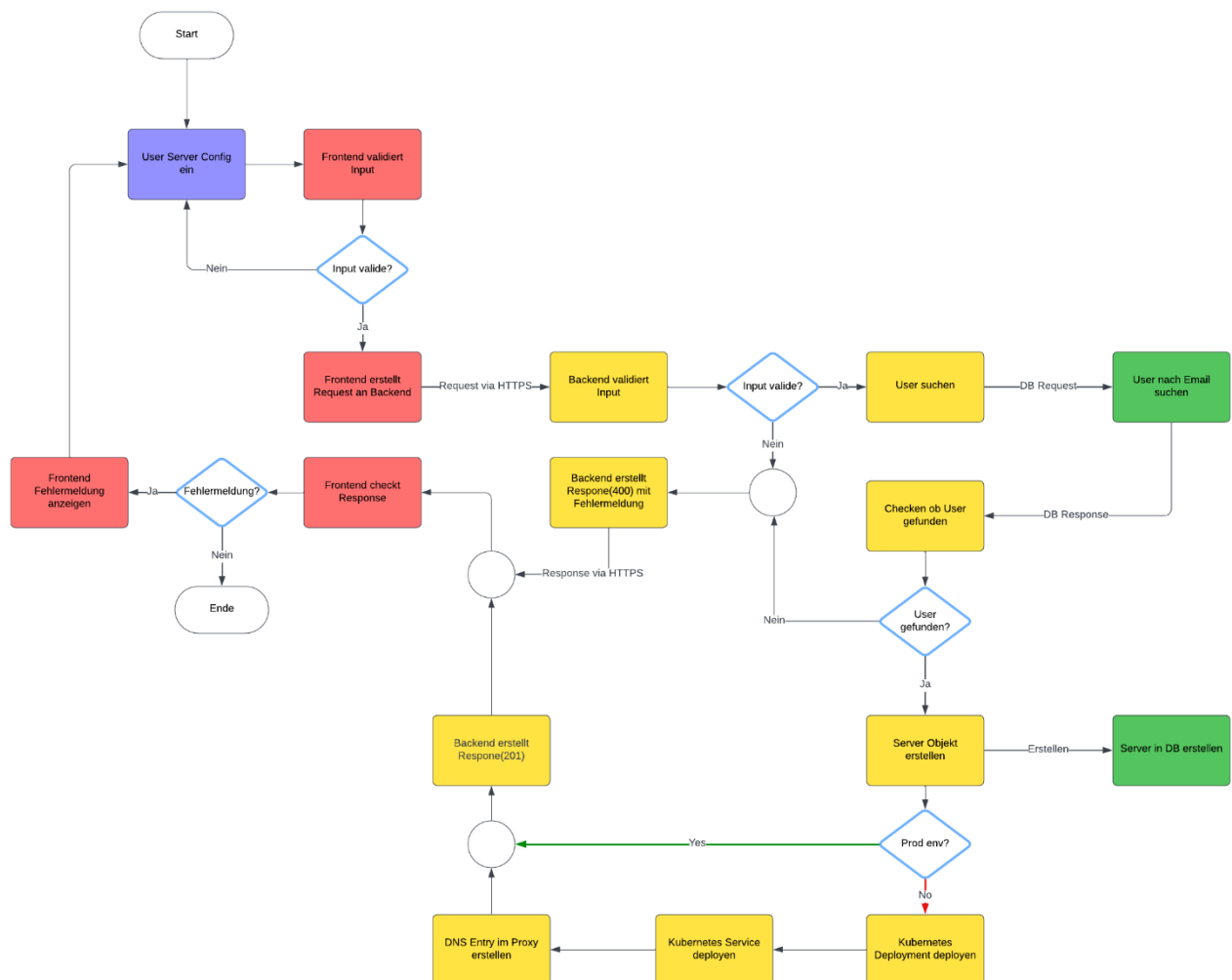
2.1.3.2 Sign in

Bei Sign In geht es um das Login. Also wenn man bereits einen Account hat und sich erneut anmelden möchte. Der Prozess ist sehr ähnlich wie beim Sign Up mit einigen kleinen Unterschieden. Denn der Prozess startet nun, wenn der User seine E-Mail sowie Passwort eingibt und das ganze vom Frontend validiert wird. Auch hier beginnt der Prozess von vorne, falls der Input nicht korrekt ist. Das Ganze wird auch hier ans Backend geschickt und dort nochmals validiert. Zudem wird hier nach einem bestehenden User in der DB gesucht und falls dieser nicht vorhanden ist, wird eine Fehlermeldung ausgelöst. Falls der User vorhanden ist, wird das Passwort mithilfe von bcrypt geprüft und wenn alles korrekt wird das Token zurück ans Backend geschickt. Das Frontend checkt wieder ob eine Fehlermeldung vom Backend kommt und falls nicht wird das Token gespeichert und der Prozess endet.



2.1.3.3 Server erstellen

Das Herzstück sowie die komplexeste Task, welche in unserer App verbaut ist das Erstellen von Servern. Der Prozess beginnt damit, dass der User im Frontend einen Server erstellt. Dessen Eingaben zuerst vom Frontend und danach vom Backend validiert. Zudem wird noch der User aus der DB geladen, damit man eine Relation schaffen kann zwischen dem Server und dem User. Danach wird das Server Objekt in der DB gespeichert. Nun wird auf das Environment geprüft. Aus Kostengründen wird nur im Prod Environment ein Server im Cluster erstellt. Falls man im Prod Environment ist, wird zuerst das Deployment und danach der Service deployed. Nun wird noch ein DNS Entry in Traefik (Proxy) erstellt, damit man auf die Gameserver auch zugreifen kann. Das Backend gibt nun eine Response entweder mit einem Error in der Validierung oder eine leere, aber erfolgreiche Response. Wenn die Response einen Fehler beinhaltet, zeigt das Frontend diesen dem User an und der Prozess beginnt von vorne. Falls alles gut läuft, ist der Prozess beendet.



2.2 Schnittstellendefinitionen

2.2.1 Frontend → Backend

Endpoint	Beschreibung	Request Headers	Request Body	Response	
POST /auth/signup	Neuen Account erstellen	n/a	<pre>{ "email": "string", "password": "string" }</pre>	201 Created	<pre>{ "token": "string" // JWT token }</pre>
				400 Bad Request	<pre>{ "error": "Invalid email format" }</pre>
				400 Bad Request	<pre>{ "error": "Invalid password format" }</pre>
				409 Conflict	<pre>{ "error": "Email already in use" }</pre>
				500 Internal Server Error	<pre>{ "error": "Failed to create user" }</pre>
POST /auth/signin	Einloggen	n/a	<pre>{ "email": "string", "password": "string" }</pre>	200 OK	<pre>{ "token": "string" // JWT token }</pre>
				400 Bad Request	<pre>{ "error": "Email and password are required" }</pre>
				401 Unauthorized	<pre>{ "error": "Invalid password or username" }</pre>
				500 Internal Server Error	<pre>{ "error": "Failed to log in" }</pre>
GET /auth/validate-token	Token validieren	<pre>{ "Authorization": "Bearer <token>" }</pre>	n/a	200 OK	<pre>{ "message": "Token is valid", "user": { "email": "string" // User's email } }</pre>
				401 Unauthorized	n/a
POST /server	Server erstellen	<pre>{ "Authorization": "Bearer <token>" }</pre>	<pre>{ "name": "string", "game": "string" }</pre>	201 Created	<pre>{ "message": "Server created successfully" }</pre>
				400 Bad Request	<pre>{ "error": "Wrong config for server" }</pre>
				400 Bad Request	<pre>{ "error": "User does not exist" }</pre>
				410 Gone	<pre>{ "error": "Server in db created but not playable due to test env" }</pre>
				500 Internal Server Error	<pre>{ "error": "Failed to create server" }</pre>
				500 Internal Server Error	<pre>{ "error": "Failed to deploy server" }</pre>

GET /server	Alle Server von einem User laden	<pre>"Authorization": "Bearer <token>"</pre>	n/a	200 OK	<pre>{ "id": "string", "name": "string", "game": "string", "url": "string" }</pre>
				400 Bad Request	<pre>{ "error": "User does not exist" }</pre>
				500 Internal Server Error	<pre>{ "error": "Failed to fetch servers" }</pre>
DELETE /server/:id	Einen server löschen	<pre>"Authorization": "Bearer <token>"</pre>	n/a	204 No Content	<pre>{ "message": "Deleted server successfully" }</pre>
				400 Bad Request	<pre>{ "error": "Failed to delete server: server does not exist or not belong to you" }</pre>
				400 Bad Request	<pre>{ "error": "User does not exist" }</pre>
				500 Internal Server Error	<pre>{ "error": "Failed to delete server" }</pre>

2.2.2 DB

Wie man mit der MongoDB mithilfe von einem MongoDB Client interagieren kann, ist in der offiziellen Dokumentation bereits dokumentiert. Folgender Link führt zu Dokumentation: <https://www.mongodb.com/>.

2.2.3 Cluster

xServer interagiert mit einem Kubernetes-Cluster, um Game-Server bereitzustellen und zu verwalten. Diese Schnittstelle wird über den ServerController im Backend implementiert. Hier sind die Details zu den relevanten Methoden und deren Schnittstellenbeschreibung.

2.2.3.1 Konfiguration

Die Verbindung zum Kubernetes-Cluster erfolgt über die Kubernetes-Client-Bibliothek `@kubernetes/client-node`. Die Konfiguration wird aus den Standard-Kubernetes-Konfigurationsdateien geladen.

```
const k8s = require('@kubernetes/client-node');
const kc = new k8s.KubeConfig();
kc.loadFromDefault();
const k8sAppsV1Api = kc.makeApiClient(k8s.AppsV1Api);
const k8sCoreV1Api = kc.makeApiClient(k8s.CoreV1Api);
```

2.2.3.2 Deployment

Ein Deployment beschreibt den gewünschten Zustand einer Anwendung und stellt sicher, dass eine spezifizierte Anzahl von Pod-Replikaten läuft. Um ein Deployment zu erstellen kann man dies folgendermassen mit der oben gezeigten AppsAPI machen:

```
await k8sAppsV1Api.createNamespacedDeployment(namespace, deploymentManifest);
```

Hierbei wird der Namespace und ein Deploymentobjekt angegeben. Der namespace ist in unserem fall «gameserver» und ein Deploymentobjekt kann folgendermassen aussehen:

```
{
  apiVersion: 'apps/v1',
  kind: 'Deployment',
  metadata: {
    name: 'mc-server',
    namespace: 'gameserver'
  },
  spec: {
    replicas: 1,
    selector: {
      matchLabels: {
        app: id
      }
    },
    template: {
      metadata: {
        labels: {
          app: 'mc-server'
        }
      }
    }
  }
},
```

```
spec: {
  containers: [{
    name: id,
    image: 'itzg/minecraft-server:latest',
    env: [{
      name: 'EULA',
      value: 'TRUE'
    },
    {
      name: "VERSION",
      value: "1.20.4"
    }
  ]],
  ports: [{
    containerPort: 25565
  }]
}]
}
```

2.2.3.3 Service

Ein Service beschreibt, wie ein Satz von Pods angesprochen werden kann und definiert eine feste IP-Adresse sowie einen DNS-Namen für eine Gruppe von Pods. Um ein Deployment zu erstellen kann man dies folgendermassen mit der oben gezeigten CoreAPI machen:

```
await k8sCoreV1Api.createNamespacedService(namespace, serviceManifest);
```

Hierbei wird der Namespace und ein Serviceobjekt angegeben. Der namespace ist in unserem fall «gameserver» und ein Serviceobjekt kann folgendermassen aussehen:

```
{
  apiVersion: 'v1',
  kind: 'Service',
  metadata: {
    name: id,
    namespace: 'gameserver'
  },
  spec: {
    selector: {
      app: 'mc-server'
    },
    ports: [{
```

```
    protocol: 'TCP',  
    port: 25565,  
    targetPort: 25565  
  }],  
  type: 'NodePort'  
}
```

2.3 Datenmodell

Da wir bis zum jetzigen Zeitpunkt viel Zeit in die Hauptfunktionalität unserer App gesteckt haben ist unser Datenmodell noch nicht so weit angewachsen. Im Moment verfügen wir unserer Mongo DB über 2 Entitäten, welchen eine Beziehung zueinander haben. Die «user» Table beinhaltet den User mit seiner E-Mail-Adresse, welche er zum Einloggen benötigt. Zudem wir auch das Passwort gespeichert, dieses ist jedoch gehasht. Als Primarykey dient eine id. Dann gibt es noch die Server Table in welcher die Gameserver der User gespeichert werden. Primarykey ist auch hier wieder eine Id. Zudem wir noch der vom User gegebene Name und das ausgewählte Game gespeichert. Eine URL wir auch gespeichert da dies die URL ist, mit welcher man auf den Gameserver connecten kann. Es gibt eine 1:n Beziehung zwischen dem User und dem Server denn ein User kann mehrere Server haben.



Da wir nicht direkt objektorientiert programmieren gibt es leider kein Klassendiagramm.

2.4 Sicherheit

2.4.1 DB

Unsere DB befindet sich in einem isolierten Netzwerk, welches durch eine Firewall geschützt ist. Dies ist auch im Systemdesign bereits erklärt und worden. Zudem ist die DB noch in einem Cluster, wobei kein Portmapping von dem Port des DB-Service zu ausserhalb des Clusters besteht. Somit können nur Services welche sich im selben Namespace wie die DB befinden (Frontend & Backend) auf die DB zugreifen. Da noch ein Passwort benötigt wird, um auf die DB zuzugreifen welches nur das Backend durch ein Secret im Cluster kennt hat nur das Backend Zugriff auf unsere DB. Dieser Zugriff findet mit der mongodb Library statt welche zum aktuellen Wissensstand sicher gegen Injections ist. Alles Passwörter welche in unsere DB geschrieben werden wurden vorher durch bcrypt mit Salt gehasht.

2.4.2 Kommunikation Frontend ↔ Backend

Die Kommunikation zwischen dem Front- und Backend finden nur über HTTPS statt. Dabei werden SSL - Zertifikate von Let's Encrypt welche sich jährlich erneuern für all unsere Domains und Subdomains verwendet.

2.4.3 JWT

Wir verwenden für die Authentifizierung JWT-Tokens und mit diesen stellen wir sicher, dass nur eingeloggte User gewisse Tätigkeiten machen können. Bei jedem Aufruf einer restriktierten Page wir im Backend das Token validiert. Zudem auch bei allen restriktierten Endpoints mithilfe einer selbstgeschriebenen Auth-Middleware.

2.4.4 Disclaimer

Da es offiziell keine Garantie auf Datenschutz gibt sind bei den Eingabefelder Warnungen und zudem befindet sich auf jeder Page eine Warnung, dass die Seite noch unter Konstruktion ist. Auch wenn wir uns ziemlich sicher sind, dass alles funktioniert, übernehmen wir keine Verantwortung.

3 Systemdokumentation

3.1 Konfigurations-Dokumentation

Für xServer gibt es drei verschiedene Konfigurationen: Development, Local und Production. Jede dieser Konfigurationen erfordert bestimmte Schritte und Tools zur Einrichtung und Ausführung des Systems. Im Folgenden werden die notwendigen Schritte und Anforderungen für jede Konfiguration beschrieben.

3.1.1 Development-Konfiguration

Die Development-Konfiguration wird verwendet, um das System in einer Entwicklungsumgebung mit allen Diensten und Abhängigkeiten vollständig auszuführen.

Allgemeine Voraussetzungen

Unabhängig von der spezifischen Konfiguration werden die folgenden Tools benötigt:

1. Docker Compose
2. npm (Node Package Manager)
3. Node.js
4. Angular CLI

DB (Docker Compose): Kommentieren Sie im **docker-compose.yaml** auf Root-Ebene alle Dienste ausser dem db-service aus. Starten Sie nun das docker compose mit «docker-compose up -d».

Frontend: Navigieren Sie zum Frontend-Ordner (./frontend), installieren Sie die Abhängigkeiten mit «npm i» und starten Sie den Entwicklungsserver mit «ng serve».

Backend: Navigieren Sie zum Backend-Ordner (./backend), installieren Sie die Abhängigkeiten mit «npm i» und starten Sie den Server mit «node ./server.js».

Nun kann man lokal entwickeln und Änderungen an dem Frontend werden automatisch übernommen und die Backend Änderungen kann man mit erneutem «node ./server.js» übernehmen. Das Frontend ist auf localhost:4200 verfügbar. Das Backend ist auf localhost:3001 verfügbar und die DB ist auf localhost:27017 verfügbar.

3.1.2 Lokale Konfiguration

Die lokale Konfiguration wird verwendet, um das System lokal mit allen Diensten und Abhängigkeiten vollständig auszuführen. Dabei lässt sich jedoch nicht entwickeln.

Allgemeine Voraussetzungen

Unabhängig von der spezifischen Konfiguration werden die folgenden Tools benötigt:

5. Docker Compose

Frontend, Backend und DB: Man kann alle 3 Komponenten ganz einfach starten indem man im root-Verzeichnis des Projekts ein Terminal öffnet und «docker-compose up -d» ausführt. Dabei sollten die Images von unserem öffentliche Container Registry gezogen werden und danach ausgeführt werden.

Das Frontend ist auf localhost:3000 verfügbar. Das Backend ist auf localhost:3001 verfügbar und die DB ist auf localhost:27017 verfügbar.

3.1.3 Prod Konfiguration

Die prod Konfiguration wird verwendet, um das System produktiv mit allen Diensten und Abhängigkeiten vollständig auszuführen auf unserem Server auszuführen. Dabei lässt sich jedoch nicht entwickeln.

Allgemeine Voraussetzungen

Unabhängig von der spezifischen Konfiguration werden die folgenden Tools benötigt:

6. Wireguard

Damit es sich überhaupt lohnt diesen Schritt zu machen braucht man eine neue Prod Version. Dafür muss man in den Deploymentfiles welche sich in «./backend/deployment», «./frontend/deployment» und «./db/deployment» befinden die Version des Images an und pusht diese Änderungen in unser Repository. Danach erstellt man ein neues Tag mit welches den Wert der Version des vorher angegebenen Docker-Images. Wichtig ist dabei dass das Tag «prod» enthalten muss. Ansonsten erkennt unsere Pipeline nicht dass es sich um einen Prod-Build handelt.

Damit man überhaupt etwas machen kann muss man sich mit Wireguard in das Netzwerk unseres Servers verbinden. Danach kann man sich mit «ssh vmadmin@192.1681.141» auf unsern Server verbinden. Zum Schutz unserer Daten geben wir das Password des Hosts sowie die VPN-Verbindung nicht heraus.

Auf diesem host läuft ein Kubernetes Cluster und es sind bereits Tools wie kubectl und git installiert. Mit «cd xserver» kann man in das xServer-Verzeichnis navigieren. Nun kann man mit «git pull» den neusten Stand unseres Repositories pullen.

DB: Falls es DB-Änderungen gibt, navigiert man in das Verzeichnis «./db/deployment» und für folgenden Befehl aus: «kubectl apply -f db-deployment-yaml».

Frontend: Falls es Frontend-Änderungen gibt, navigiert man in das Verzeichnis «./frontend/deployment» und für folgenden Befehl aus: «kubectl apply -f frontend-deployment-yaml».

Backend: Falls es Backend-Änderungen gibt, navigiert man in das Verzeichnis «./backend/deployment» und für folgenden Befehl aus: «kubectl apply -f backend-deployment-yaml».

Innerhalb einiger Sekunden sollten die neuen Versionen installiert sein und man diese über «host.xserver.space» betrachten.

3.2 Benutzerhandbuch

3.2.1 Systemübersicht

Ziele des Systems

Dieses Benutzerhandbuch bietet eine umfassende Übersicht über die xServer-Plattform, ihre Ziele und Hauptfunktionen. Es richtet sich an Anwender, die die Plattform zur Verwaltung und Bereitstellung von Gameservern nutzen möchten.

Ziele des Systems

Die xServer-Plattform wurde entwickelt, um Einzelpersonen und kleinen Gaming-Communities eine benutzerfreundliche und effiziente Möglichkeit zu bieten, eigene Gameserver zu konfigurieren, zu verwalten und zu betreiben. Die Hauptziele des Systems sind:

1. **Benutzerfreundlichkeit:** Bereitstellung einer intuitiven Oberfläche, die es auch technisch weniger versierten Nutzern ermöglicht, Gameserver einfach zu erstellen und zu verwalten.
2. **Automatisierung:** Reduzierung des manuellen Aufwands durch automatisierte Bereitstellung und Konfiguration von Servern.
3. **Flexibilität:** Ermöglichung einer flexiblen Anpassung der Serverressourcen und -einstellungen entsprechend den individuellen Bedürfnissen der Nutzer.
4. **Sicherheit:** Gewährleistung der Datensicherheit und des Datenschutzes durch robuste Sicherheitsmassnahmen und Compliance mit gesetzlichen Anforderungen.

Struktur des Systems und externe Schnittstellen

Das System besteht aus 3 Komponenten dem Frontend welches über host.xserver.space erreichbar ist und für alle Nutzer zugänglich ist. Ein Backend welches über backend.xserver.space zugänglich ist jedoch für die meisten Endpoint ein Login benötigt wird. Zuletzt gibt es noch eine DB, welche jedoch von externem Zugriff geschützt ist.

Allgemeines zu Sicherheit, Datenschutz, Anwenderrollen

Anwenderrollen:

Es gibt in der aktuellen Version von xServer keine verschiedenen Rollen. Jeder Nutzer ist gleichwertig und kann sich einen Account erstellen, mit welchem er sich Server erstellen kann.

Datenschutz

Wie bereits oben genannt befindet sich unsere DB hinter einer Firewall in einem privaten Netzwerk, welches von ausserhalb keine DB-Connection zulässt. In dieses Netzwerk gibt es nur 2 verschiedene VPN-Connections, welche im Besitz von den Inhabern von xServer Maurice Däppen und Patrick Aeschlimann sind. Des Weiteren befindet sich die DB in einem Kubernetes Cluster, welcher in sich ein weiteres Netzwerk hat wo nochmals keine DB-Connections erlaubt sind. Dadurch kann nur das Backend auf die DB zugreifen.

Alle Daten werden über HTTPS übertragen was die Daten bei der Datenübertragung schützt. Alle Passwörter welche in unserer DB sind, sind mithilfe bcrypt mit Salt gehasht.

Sicherheit

Wie im Kapitel Datenschutz gesagt befindet sich unsere DB hinter einer Firewall und mit gesichertem Zugriff. Da unser Backend den MongoDB Client verwendet sind keine Injektion möglich. Des weiteren wird alles mit HTTPS übertragen.

3.2.2 Anwenderfunktionalität

F1 Sign Up:

Damit man xServer richtig nutzen kann benötigt man einen Account deshalb ist hier die Anleitung dazu.

Vorbedingungen: Nicht eingeloggt sein.

- Besuche <https://host.xserver.space/signup> .
- Gib dort deine E-Mail-Adresse ein.
- Gib ein Passwort mit mind. 6 Zeichen ein.
- Bestätige dein Passwort.
- Drücke «Sign Up»

Falls eine Fehlermeldung erscheint, schau im Anhang des Supporthandbuches an was zu machen ist. Falls alles gut verläuft, solltest du auf die Startseite weitergeleitet werden und oben rechts sollte nun «Sign out» stehen.

F2 Sign In:

Damit man xServer richtig nutzen kann muss man sich auch in seinen Account einloggen deshalb ist hier die Anleitung dazu.

Vorbedingungen: Nicht eingeloggt sein und einen Account besitzen.

- Besuche <https://host.xserver.space/signin> .
- Gib dort deine E-Mail-Adresse ein.
- Gib dein Passwort ein.
- Drücke «Sign In»

Falls eine Fehlermeldung erscheint, schau im Anhang des Supporthandbuches an was zu machen ist. Falls alles gut verläuft, solltest du auf die Startseite weitergeleitet werden und oben rechts sollte nun «Sign out» stehen.

F3 Sign Out:

Damit man xServer richtig nutzen kann muss man sich auch in seinen Account einloggen deshalb ist hier die Anleitung dazu.

Vorbedingungen: Eingeloggt sein (F1 oder F2).

- Besuche <https://host.xserver.space/> .
- Drücke oben rechts «Sign out»
- Drücke «Sign In»

Falls eine Fehlermeldung erscheint, schau im Anhang des Supporthandbuches an was zu machen ist. Falls alles gut verläuft, solltest du auf die Startseite weitergeleitet werden und oben rechts sollte nun «Sign In» und «Sign up» stehen. Zudem solltest du unten in der Mitte ein Toast sehen mit der Meldung «Sign out succeeded».


F4 Server erstellen:

Die Hauptfunktionalität von xServer ist natürlich server zu erstellen. Hier kommt die Anleitung.

Vorbedingungen: Eingeloggt sein (F1 oder F2).

1. Besuche <https://host.xserver.space/servers> .
2. Drücke auf den Button «Create Server»
3. Gib einen Namen für deinen Server ein.
4. Wähle ein verfügbares Spiel aus.
5. Drücke «Create»

Falls eine Fehlermeldung erscheint, schau im Anhang des Supporthandbuches an was zu machen ist. Falls alles gut verläuft, sollte nun ein Server in der Liste angezeigt werden:

Name	URL	Game	RAM	CPU	
My-TestServer	mc-c00d7c9f-2826-4e4d-b96d-bf82b559dda.xserver.space	MINECRAFT	n/a	n/a	

F5 Server löschen:

Wenn man Server erstellt will man diese im Zweifelsfall auch wieder löschen. Hier ist die Anleitung.

Vorbedingungen: Eingeloggt sein (F1 oder F2) und einen Server erstellt haben (F4).

1. Besuche <https://host.xserver.space/servers> .
2. Drücke auf den Button mit dem Mülleimer Symbol:



Falls eine Fehlermeldung erscheint, schau im Anhang des Supporthandbuches an was zu machen ist. Falls alles gut verläuft, sollte nun der ausgewählte Server nicht mehr in der Liste sein.

3.3 Integrations- und Installationshandbuch

Im Kapitel der Konfigurations-Dokumentation sowie in den README's der einzelnen Projekten wir für Fachpersonen erklärt wie sie unser System installieren kann. Für alle anderen Anwender muss nichts installiert werden da unsere Applikation im Browser läuft. Dafür wird nur ein Browser benötigt welcher JavaScript Enabled hat. Wie der User seine Gameclients installiert und konfiguriert ist nicht in unserer Verantwortung. Man kann einfach «host.xserver.space» besuchen und versuchen einen Account zu erstellen oder sich anzumelden wenn dies Funktioniert sollte das System ohne Probleme laufen.

3.4 Supporthandbuch

Wir bieten keinen direkten(betreuten) Usersupport an falls es jedoch Problem und Fragen kann man sich an «p.a.tu1996 (at) hotmail.com» was auch auf der Website genannt wird. Supportanfragen via diese E-Mail werden innerhalb von 5 bis 10 Werktagen bearbeitet.

3.4.1 Massnahmen bei Benutzerproblemen

Das sich das System noch in Entwicklung befindet, was auch deutlich angegeben ist, bitten wir User um Verständnis bei Problemen und bitten sie einige Tage zu warten und danach nochmals zu versuchen. Falls das Problem immer noch besteht, sollte eine Mail an die angegebene E-Mail-Adresse geschickt werden.

Falls man den Fehler «Wrong config for server» erhält liegt das daran, dass man dem Server einen kryptischen Namen geben muss, da wir bis jetzt kein Zahlungssystem eingebaut haben und wir das unnötige(kostenlose) deployen von Ressourcen unterdrücken möchten. Falls du als User den geforderten Namen nicht kennst bist du im Moment auch nicht dazu berechtigt einen Server zu erstellen.

3.4.2 Massnahmen bei technischen Problemen

Bei technischem Problem oder nicht Erreichbarkeit der Website oder von Gameserver bitten wir den User einige Tage zu warten und nochmals zu versuchen da sich xServer immer noch ein der Entwicklung befindet.

3.4.3 Anhang zum Supporthandbuch

Fehlermeldungen:

Fehlermeldung	Bedeutung	Problembehebung
---------------	-----------	-----------------

Invalid email format	E-Mail-Format nicht korrekt	E-Mail im Format xxx@xx.xx eingeben
Invalid password format	Passwort Richtlinien nicht eingehalten	Passwort mit mindestens 6 Zeichen eingeben
Email already in use	Angegebene E-Mail wird bereits von einem anderen User verwendet	Eine Andere E-Mail wählen
Failed to create user	Account konnte nicht erstellt werden	Später nochmals versuchen
Email and password are required	Email oder Passwort beim Login nicht angegeben	Email und Passwort beim Login angeben
Invalid password or username	Ungültige Logindaten	Überprüfen und nochmals angeben.
Failed to log in	Fehler beim Login Prozess	Später nochmals versuchen
User does not exist	Der Angemeldete Nutzer stimmt nicht mit dem Token überein	Abmelden und nochmals anmelden
Failed to create server	Fehler bei erstellen des Servers	Später nochmals versuchen
Failed to deploy server	Server konnte nicht erstellt werden	Später nochmals versuchen
Server in db created but not playable due to test env	Server wurde nur in der DB und nicht als Ressource angelegt aufgrund von Falscher Umgebung	URL überprüfen
Failed to fetch servers	Fehler bei laden aller Server	Später nochmals versuchen
Failed to delete server: server does not exist or not belong to you	Server existiert nicht oder gehört nicht dir	Nicht mehr machen
Failed to delete server	Fehler bei Server löschen	Später nochmals versuchen

4 Test

4.1 Unit Test

Für die Entwicklung und Überprüfung der Funktionalität von xServer setzen wir Unittests ein, die mit Jest und ts-mockito durchgeführt werden. Diese Tools ermöglichen es uns, unsere TypeScript-Komponenten effektiv zu testen und sicherzustellen, dass jede Funktion wie erwartet arbeitet.

Unittests werden in Dateien mit der Endung «.spec.ts» geschrieben, um sie klar als Testdateien zu kennzeichnen. Ein Beispiel für einen Unittest ist im Anhang B zu finden.

Im Beispiel im Anhang wird ein Mock-Objekt für StoreService erstellt und in den SignupContainer injiziert. Der Test überprüft, ob die signup-Methode die dispatch-Methode von StoreService aufruft, wenn sie mit einem SignUpDto-Objekt aufgerufen wird. Solche Tests helfen, die Integrität und Zuverlässigkeit des Codes sicherzustellen.

Durch die Verwendung von Jest und ts-mockito können wir sicherstellen, dass unsere Komponenten isoliert getestet werden und jede Abhängigkeit korrekt simuliert wird. Dies ermöglicht eine präzise und effiziente Fehlererkennung und -behebung während der Entwicklung.

4.2 Systemtest - Testspezifikation

4.2.1 Kritikalität der Funktionseinheit

Funktionseinheit	Risiko (klein, mittel, hoch)	Auswirkungen (klein, mittel, hoch)	Folgen
Frontend	klein	hoch	<ol style="list-style-type: none"> Verlust der Kunden. Negative SEO-Auswirkung

Backend	klein	hoch	<ol style="list-style-type: none"> 1. Evtl. Datenverlust. 2. Keine Funktionalität. 3. Kritisch für den Umsatz.
Cluster	klein	hoch	<ol style="list-style-type: none"> 1. Totalausfall. 2. Alles Komplett unerreichbar. 3. Unmöglich bereitstellen von jeglichen Diensten.
Datenbank	klein	hoch	<ol style="list-style-type: none"> 1. Datenverlust 2. Ausfall der meisten Dienste 3. Login unmöglich
Login	klein	hoch	Für Kunden unmöglich jegliche Funktionen zu nutzen.
Sign-up	klein	mittel	<ul style="list-style-type: none"> - Keine neue Gewinnung von Kunden. - Schlechte Benutz Erfahrung. - Umsatzverlust.
Proxy Host	klein	hoch	<ul style="list-style-type: none"> - Sämtliche Dienste nicht mehr erreichbar. - Kein SSL-Zertifikat mehr möglich.

4.2.2 Testanforderungen

In diesem Abschnitt werden die Testanforderungen für xServer beschrieben. Dabei wird zwischen verschiedenen Testkategorien unterschieden, um sicherzustellen, dass alle Aspekte des Systems gründlich überprüft werden. Funktionale Tests dienen der Überprüfung, ob alle spezifizierten Funktionen korrekt implementiert und funktionsfähig sind. Integrationstests stellen sicher, dass alle Komponenten, wie Kubernetes, Docker-Container und Proxymanager, nahtlos zusammenarbeiten. Schliesslich werden Systemtests durchgeführt, um das gesamte System in einer produktionsähnlichen Umgebung zu überprüfen.

- **Funktionale Tests:**
Überprüfung, ob alle spezifizierten Funktionen korrekt implementiert und funktionsfähig sind.
- **Integrationstests:** Sicherstellen, dass alle Komponenten (Kubernetes, Docker-Container, Proxymanager) nahtlos zusammenarbeiten.
- **Systemtests:**
Überprüfung des gesamten Systems in einer produktionsähnlichen Umgebung.

4.2.3 Testverfahren

Funktionale Tests

Ziel: Überprüfung, ob alle spezifizierten Funktionen korrekt implementiert und funktionsfähig sind.

Vorgehen:

Anforderungen analysieren: Erstellen Sie eine detaillierte Liste aller funktionalen Anforderungen.

Testfälle entwickeln: Schreiben Sie spezifische Testfälle für jede Anforderung. Nutzen Sie Techniken wie Äquivalenzklassenbildung und Grenzwertanalyse.

Automatisierung: Implementieren Sie automatisierte Tests (z.B. mit Tools wie Selenium für Web-Interfaces oder Postman für API-Tests), um die Testeffizienz zu erhöhen.

Manuelle Tests: Ergänzen Sie die automatisierten Tests durch manuelle Tests, um komplexe Anwendungsfälle und Benutzerinteraktionen abzudecken.

Integrationstests

Ziel: Sicherstellen, dass alle Komponenten (Kubernetes, Docker-Container, Proxymanager) nahtlos zusammenarbeiten.

Vorgehen:

Testumgebung einrichten: Stellen Sie eine Testumgebung bereit, die alle Komponenten enthält und die reale Produktionsumgebung so genau wie möglich simuliert.

Integrationstestfälle: Entwickeln Sie Testfälle, die die Interaktion zwischen den verschiedenen Komponenten überprüfen. Achten Sie darauf, typische Szenarien und Fehlerfälle abzudecken.

CI/CD-Pipeline: Integrieren Sie die Integrationstests in die CI/CD-Pipeline, um sicherzustellen, dass alle Änderungen automatisch getestet werden.

Systemtests

Ziel: Überprüfung des gesamten Systems in einer produktionsähnlichen Umgebung.

Vorgehen:

End-to-End-Tests: Entwickeln Sie umfassende End-to-End-Testszenarien, die das gesamte System abdecken, von der Benutzereingabe bis zur Ausgabe.

Testdaten: Erstellen Sie realistische Testdaten, die die tatsächlichen Daten im Produktivbetrieb simulieren.

Produktionsähnliche Umgebung: Stellen Sie sicher, dass die Testumgebung die Produktionsumgebung so genau wie möglich nachbildet, einschliesslich Hardware, Netzwerkkonfiguration und Softwareversionen.

Lasttests: Führen Sie Lasttests durch, um zu überprüfen, wie das System unter realen Betriebsbedingungen reagiert.

Testverfahren

Vorbereitung:

Testplan erstellen: Definieren Sie den Umfang, die Ziele und den Zeitplan der Tests.

Testdaten generieren: Erstellen oder beschaffen Sie die notwendigen Testdaten.

Testumgebung aufbauen: Richten Sie die erforderlichen Testumgebungen ein, einschliesslich aller notwendigen Hardware und Software.

Durchführung:

Tests ausführen: Führen Sie die Tests gemäss dem Testplan durch.

Ergebnisse dokumentieren: Protokollieren Sie die Testergebnisse sorgfältig und dokumentieren Sie aufgetretene Fehler oder Abweichungen.

Auswertung:

Ergebnisse analysieren: Analysieren Sie die Testergebnisse, um Schwachstellen und Fehlerquellen zu identifizieren.

Berichte erstellen: Erstellen Sie detaillierte Testberichte, die die Ergebnisse und Empfehlungen enthalten.

Korrekturmassnahmen: Planen und implementieren Sie erforderliche Korrekturmassnahmen, um identifizierte Probleme zu beheben.

Durch die systematische Anwendung dieser Vorgehensweise können Sie sicherstellen, dass alle Aspekte von xServer gründlich getestet werden und das System die gestellten Anforderungen erfüllt.

4.2.4 Testkriterien

Abdeckungsgrad:

Die Testabdeckung umfasst alle Hauptfunktionen und -komponenten von xServer, einschliesslich der Registrierung und Erstellung von Gameservern sowie der Funktionalität des Proxymanagers.

Sowohl positive als auch negative Testfälle sollten berücksichtigt werden, um sicherzustellen, dass das System unter verschiedenen Bedingungen stabil und fehlerfrei arbeitet.

Die Testabdeckung erstreckt sich auf verschiedene Betriebsbedingungen, einschliesslich Normalbetrieb, Höchstleistungszenarien und Komponentenausfällen.

4.2.4.1 Checklisten

Funktionale Checkliste:

- Registrierung neuer Benutzer
- Erstellung eines Gameservers
- Abrechnung und Zahlungsabwicklung
- Lastverteilung und Skalierung durch den Proxymanager
- Sicherheitsüberprüfungen und Zugriffskontrollen

Nicht-funktionale Checkliste:

- Performance-Tests (Reaktionszeiten, Durchsatzraten)
- Sicherheitstests (Authentifizierung, Verschlüsselung)
- Usability-Tests (Benutzerfreundlichkeit der Website, Responsivität)
- Skalierbarkeitstests (Verhalten unter Last)
- Wiederherstellungstests nach Komponentenausfällen

Integrationstests-Checkliste:

- Korrekte Interaktion zwischen Kubernetes, Docker-Containern und dem Proxymanager
- Datenkonsistenz und Kommunikation zwischen den Komponenten
- Erfolgreiche Integration von Zahlungs- und Abrechnungsdiensten

4.2.4.2 Endkriterien

Der Test wird als erfolgreich abgeschlossen betrachtet, wenn alle funktionalen und nicht-funktionalen Anforderungen erfüllt sind und die Systemleistung unter Normal- und Ausnahmebedingungen zufriedenstellend ist.

Die Registrierung und Erstellung von Gameservern erfolgt ohne Fehler und Verzögerungen, und die Zahlungsabwicklung verläuft reibungslos.

Das System zeigt eine hohe Skalierbarkeit und Stabilität, selbst unter hoher Last oder bei Ausfall von Komponenten.

Sicherheitsaspekte wie Datenverschlüsselung und Zugriffskontrollen sind effektiv implementiert und bieten angemessenen Schutz vor unbefugtem Zugriff.

Die Benutzerfreundlichkeit der Website ist hoch, und die Benutzeroberfläche ist intuitiv gestaltet und reagiert schnell auf Benutzerinteraktionen.

Diese Kriterien helfen dabei, den Testprozess zu strukturieren und sicherzustellen, dass xServer gründlich und effektiv getestet wird, um eine hohe Qualität und Zuverlässigkeit des Systems zu gewährleisten.

4.2.5 Testfälle

Hier ist eine Übersicht der Testfälle und der damit überprüften Anforderungen für xServer, einschliesslich einer Tabelle mit Testfällen. Dies sind die Testfälle welche bis jetzt durchgeführt werden können da das System noch nicht fertig entwickelt wurde:

Testfall	User Story	Überprüfte Anforderungen
Registrierung neuer Benutzer	Als potenzieller Benutzer möchte ich mich auf der xServer-Website registrieren, um einen Account zu erstellen und Gameserver zu hosten.	Benutzer sollten sich erfolgreich registrieren können Erfassung und Speicherung von Benutzerdaten Fehlerbehandlung bei ungültigen Registrierungsversuchen.
Anmeldung mit einem Benutzer	Als potenzieller Benutzer möchte ich mich auf der xServer-Website bei meinem Account anmelden können, um einen Account zu erstellen und Gameserver zu hosten.	Benutzer sollten sich erfolgreich anmelden können. Fehlerbehandlung bei ungültigen Anmeldeversuchen.
Erstellung eines Gameservers	Als registrierter Benutzer möchte ich über xServer einen Gameserver erstellen, um mein Spiel zu hosten.	Erfolgreiche Erstellung eines Gameservers Konfiguration und Bereitstellung des Gameservers.
Löschen eines Gameservers	Als registrierter Benutzer möchte ich über xServer einen von meinen Gameserver löschen.	Erfolgreiche Löschung eines eigenen Gameservers.

4.3 Testprozedur

4.3.1 Test 1

Beschreibung	Registrierung eines neuen Benutzers -> erfolgreich	
Abgedeckte Anwendungsfälle	Registrierung neuer Benutzer	
Ausgangssituation	Es besteht noch kein solcher Benutzer.	
Vorbereitungsschritte	<ol style="list-style-type: none"> 1. Stellen Sie sicher, dass kein Benutzer mit der gewählten E-Mail bereits existiert. 2. Starten Sie den Webbrowser. 	
Testschritte	Erwartetes Resultat	
1. Geben Sie die URL der xServer-Website (host.xserver.space) in die Adressleiste des Browsers ein.	Erwartetes Ergebnis: Die Startseite von xServer wird angezeigt.	
2. Wählen Sie die Option "Sign Up".	Erwartetes Ergebnis: Das Registrierungsformular wird angezeigt.	
3. Eingeben der folgenden Informationen: <ul style="list-style-type: none"> • E-Mail: test@test.ch • Passwort: asdfgh • Passwortwiederholung: asdfgh 	Erwartetes Ergebnis: Der Submit Button wird nicht mehr ausgegraut angezeigt.	
4. Submit drücken	Man wird auf die Startseite weitergeleitet und oben rechts sollte nun «Sign out» stehen	

4.3.2 Test 2

Beschreibung	<i>Registrierung eines neuen Benutzers -> Nicht erfolgreich (E-Mail schon in Gebrauch)</i>	
Abgedeckte Anwendungsfälle	Registrierung neuer Benutzer	
Ausgangssituation	<i>Es besteht bereits ein user mit der E-Mail-Adresse test@test.ch</i>	
Vorbereitungsschritte	<ol style="list-style-type: none"> 1. Test 1 durchführen 2. Starten Sie den Webbrowser. 	
Testschritte		Erwartetes Resultat
1. Geben Sie die URL der xServer-Website (host.xserver.space) in die Adressleiste des Browsers ein.		Erwartetes Ergebnis: Die Startseite von xServer wird angezeigt.
2. Wählen Sie die Option "Sign Up".		Erwartetes Ergebnis: Das Registrierungsformular wird angezeigt.
3. Eingeben der folgenden Informationen: <ul style="list-style-type: none"> • E-Mail: test@test.ch • Passwort: asdfgh • Passwortwiederholung: asdfgh 		Erwartetes Ergebnis: Der Submit Button wird nicht mehr ausgegraut angezeigt.
4. Submit drücken		Man bleibt auf der Registrierungsseite und ein Toast mit einer Fehlermeldung, dass die E-Mail schon in gebraucht ist wird angezeigt.

4.3.3 Test 3

Beschreibung	<i>Anmelden mit einem existierenden Benutzer</i>	
Abgedeckte Anwendungsfälle	Anmeldung mit einem Benutzer	
Ausgangssituation	<i>Es besteht bereits ein user mit der E-Mail-Adresse test@test.ch</i>	
Vorbereitungsschritte	<ol style="list-style-type: none"> 1. Test 1 durchführen 2. Starten Sie den Webbrowser. 	
Testschritte		Erwartetes Resultat
1. Geben Sie die URL der xServer-Website (host.xserver.space) in die Adressleiste des Browsers ein.		Erwartetes Ergebnis: Die Startseite von xServer wird angezeigt.
2. Wählen Sie die Option "Sign In".		Erwartetes Ergebnis: Das Anmeldeformular wird angezeigt.
3. Eingeben der folgenden Informationen: <ul style="list-style-type: none"> • E-Mail: test@test.ch • Passwort: asdfgh 		Erwartetes Ergebnis: Der Submit Button wird nicht mehr ausgegraut angezeigt.
4. Submit drücken		Man wird auf die Startseite weitergeleitet und oben rechts sollte nun «Sign out» stehen.

4.3.4 Test 4

Beschreibung	<i>Anmelden mit einem nicht existierenden Benutzer</i>	
Abgedeckte Anwendungsfälle	Anmeldung mit einem Benutzer	
Ausgangssituation	<i>Es besteht kein ein user mit der E-Mail-Adresse test2@test.ch</i>	
Vorbereitungsschritte	1. Starten Sie den Webbrowser.	
Testschritte	Erwartetes Resultat	
1. Geben Sie die URL der xServer-Website (host.xserver.space) in die Adressleiste des Browsers ein.	<i>Erwartetes Ergebnis: Die Startseite von xServer wird angezeigt.</i>	
2. Wählen Sie die Option "Sign In".	<i>Erwartetes Ergebnis: Das Anmeldeformular wird angezeigt.</i>	
3. Eingeben der folgenden Informationen: <ul style="list-style-type: none"> E-Mail: test2@test.ch Passwort: asdfgh 	<i>Erwartetes Ergebnis: Der Submit Button wird nicht mehr ausgegraut angezeigt.</i>	
4. Submit drücken	<i>Man bleibt auf der Anmeldeseite und ein Toast mit einer Fehlermeldung, dass die Daten inkorrekt sind.</i>	

4.3.5 Test 5

Beschreibung	<i>Server erstellen</i>	
Abgedeckte Anwendungsfälle	Erstellung eines Gameservers	
Ausgangssituation	<i>Es besteht ein User und man ist eingeloggt</i>	
Vorbereitungsschritte	1. Test 1 durchführen und angemeldet bleiben 2. Starten Sie den Webbrowser.	
Testschritte	Erwartetes Resultat	
1. Geben Sie die URL der xServer-Website (host.xserver.space) in die Adressleiste des Browsers ein.	<i>Erwartetes Ergebnis: Die Startseite von xServer wird angezeigt.</i>	
2. Wählen Sie die Option "Servers".	<i>Erwartetes Ergebnis: Die Server Seite wird angezeigt</i>	
3. Klicken Sie den Button «Create Server»	<i>Erwartetes Ergebnis: Ein Dialog zum Server erstellen wird geöffnet</i>	
4. Den Namen (hier nicht genannt damit man nicht unnötig ressourcen deployen kann -> bei Maurice Nachfragen) eingeben Und Minecraft auswählen	<i>Submit Button wird nicht mehr ausgegraut angezeigt.</i>	
5. Submit drücken	<i>Dialog schlisst sich und man sieht seinen Server in der Liste.</i>	

4.3.6 Test 6

Beschreibung	Server löschen	
Abgedeckte Anwendungsfälle	Löschen eines Gameservers	
Ausgangssituation	Es besteht ein User und man ist eingeloggt und man hat bereits einen Server erstellt	
Vorbereitungsschritte	1. Test 5 durchführen und angemeldet bleiben 2. Starten Sie den Webbrowser.	
Testschritte		Erwartetes Resultat
1. Geben Sie die URL der xServer-Website (host.xserver.space) in die Adressleiste des Browsers ein.		Erwartetes Ergebnis: Die Startseite von xServer wird angezeigt.
2. Wählen Sie die Option "Servers".		Erwartetes Ergebnis: Die Server Seite wird angezeigt, zudem sollte dein Server sichtbar sein
3. Klicken Sie den Button mit dem Mülleimer		Erwartetes Ergebnis: Der Server solle verschwinden

4.4 Testprotokoll

Test 1

Getestete Version: prod-0.1.5
 Tester: Fabrice Däppen (Bruder von Maurice)
 Datum, Zeit: 18.5.2024, 18.00 – 18.15

Testfall 1 „Registrierung eines neuen Benutzers -> erfolgreich“

Testschritt	Erfüllt	Bemerkung
1.	<input checked="" type="checkbox"/>	
2.	<input checked="" type="checkbox"/>	
3.	<input checked="" type="checkbox"/>	
4.	<input checked="" type="checkbox"/>	Es gab keine Abweichungen von den erwarteten Resultaten.

Test 2

Getestete Version: prod-0.1.5
 Tester: Fabrice Däppen (Bruder von Maurice)
 Datum, Zeit: 18.5.2024, 18.15 – 18.25

Testfall 2 „Registrierung eines neuen Benutzers -> Nicht erfolgreich (E-Mail schon in Gebrauch)“

Testschritt	Erfüllt	Bemerkung
1.	<input checked="" type="checkbox"/>	
2.	<input checked="" type="checkbox"/>	
3.	<input checked="" type="checkbox"/>	
4.	<input checked="" type="checkbox"/>	Wie erwartet erschie eine Fehlermeldung

Test 3

Getestete Version: prod-0.1.5
 Tester: Fabrice Däppen (Bruder von Maurice)
 Datum, Zeit: 18.5.2024, 18.30 – 18.40

Testfall 3 „Anmelden mit einem existierenden Benutzer“

Testschritt	Erfüllt	Bemerkung
1.	<input checked="" type="checkbox"/>	
2.	<input checked="" type="checkbox"/>	
3.	<input checked="" type="checkbox"/>	
4.	<input checked="" type="checkbox"/>	Login funktionierte

Test 4

Getestete Version: prod-0.1.5
Tester: Fabrice Däppen (Bruder von Maurice)
Datum, Zeit: 18.5.2024, 18.40 – 18.50

Testfall 4 „Anmelden mit einem nicht existierenden Benutzer“

Testschritt	Erfüllt	Bemerkung
1.	<input checked="" type="checkbox"/>	
2.	<input checked="" type="checkbox"/>	
3.	<input checked="" type="checkbox"/>	
4.	<input checked="" type="checkbox"/>	Fehlermeldung wurde wie erwartet angezeigt

Test 5

Getestete Version: prod-0.1.5
Tester: Fabrice Däppen (Bruder von Maurice)
Datum, Zeit: 18.5.2024, 18.50 – 19:00

Testfall 5 „Server erstellen“

Testschritt	Erfüllt	Bemerkung
1.	<input checked="" type="checkbox"/>	
2.	<input checked="" type="checkbox"/>	
3.	<input checked="" type="checkbox"/>	
4.	<input checked="" type="checkbox"/>	Musste Maurice nach speziellen Name fragen.
5.	<input checked="" type="checkbox"/>	Server konnte erstellt werden und man konnte darauf spielen

Test 6

Getestete Version: prod-0.1.5
Tester: Fabrice Däppen (Bruder von Maurice)
Datum, Zeit: 18.5.2024, 18.50 – 19:00

Testfall 6 „Server löschen“

Testschritt	Erfüllt	Bemerkung
1.	<input checked="" type="checkbox"/>	
2.	<input checked="" type="checkbox"/>	
3.	<input checked="" type="checkbox"/>	Funktionierte sehr schnell

Auswertung:

Im Rahmen der durchgeführten Tests wurde festgestellt, dass alles einwandfrei funktioniert und keine Mängel vorliegen. Abweichungen der Testresultate zu den Systemanforderungen wurden dokumentiert (gab jedoch keine) und deren Einfluss auf die Funktionstüchtigkeit des Systems bewertet (nicht nötig da keine Mängel). Es zeigte sich kein bestimmter Trend im Auftreten gleichartiger Mängel, sodass diesbezügliche Vermutungen nicht notwendig waren.

5 Antrag auf Freigabe der nächsten Projektphase

Das Projektteam empfiehlt dem Auftraggeber die Freigabe der Phase Einführung.

Anhang A

Kein Quellcode → Mit Herrn Kissling abgesprochen

Anhang B

Unittest:

```
import { SignUpDto } from './signup.model';
import { StoreService } from '../../../shared/ngrx/store.service';
import { anything, instance, mock, verify } from 'ts-mockito';
import { SignupContainer } from './signup.container';

describe('SignupContainer', () => {
  let signupContainer: SignupContainer;
  let storeServiceMock: StoreService;

  beforeEach(() => {
    storeServiceMock = mock(StoreService);
    signupContainer = new SignupContainer(instance(storeServiceMock));
  });

  it('should dispatch trySignup action when signup is called', () => {
    const dto: SignUpDto = { } as SignUpDto;

    // act
    signupContainer.signup(dto);

    // assert
    // anything da ngrx action nicht mockable sind
    verify(storeServiceMock.dispatch(anything())) .once();
  });
});
```