I- EINFÜHRUNG UND ZIELE

1. Projektbeschreibung:

Entwicklung eines hochverfügbaren und benutzerfreundlichen Krankenhaus-Managementsystems, das die Überwachung der Bettenauslastung ermöglicht.

Features:

- GUI, die dem Nutzer ermöglicht, den Zustand der Krankenhäuser bezüglich der Bettenauslastung zu sehen und zu bearbeiten.
- Automatische Aktualisierung der Daten im System (im Server).
- Das System ist erweiterbar, indem neue Krankenhäuser hinzugefügt werden können.

Hauptziel:

Das Projekt zielt darauf ab, ein zuverlässiges System zur Echtzeitüberwachung der Bettenauslastung zu schaffen. Dabei werden Krankenhäuser in die Lage versetzt, Daten präzise und zeitnah zu aktualisieren, während Administratoren und Beobachter eine benutzerfreundliche Oberfläche erhalten, um Entscheidungen bezüglich Ressourcenallokation und Notfallmanagement zu unterstützen.

2. Qualitätsziele

Qualitätsziel	Motivation und Erläuterung
Datenqualität	Die Daten auf dem Server werden nach jeder Änderung in den Krankenhäusern aktualisiert, um relevante und zeitnahe Entscheidungen zu ermöglichen.
Verfügbarkeit	Um einen unterbrechungsfreien Betriebsablauf zu gewährleisten, sollte das System nicht länger als 15 Minuten ausfallen (Failover-Mechanismen).
Skalierbarkeit	Das Hinzufügen oder Entfernen eines Krankenhauses sollte einfach sein, ohne die Leistung des Systems zu beeinträchtigen.
Benutzerfreundlichkeit	Die Bedienung des Systems soll unkompliziert sein, indem lediglich die gängigen Elemente des Frontends aufgesetzt werden, jeder mit einer klaren Beschreibung seiner Funktion (z. B. ein Knopf, auf dem seine Funktion deutlich angegeben ist).
Wartbarkeit	Der gesamte Code des Systems muss sorgfältig dokumentiert werden, und die Namen der Klassen sowie der Methoden müssen präzise auf ihre jeweiligen Aufgaben hinweisen. Zudem sollte das Atomar-Prinzip der Methoden beibehalten werden.

3. Stakeholder

wer?	Kontakt	Interesse, Bezug, Erwartungen
Krankenhaus-Mitarbeiter	Intern	- Er möchte das System über die GUI nutzen, um die aktuelle Lage in den verschiedenen Krankenhäusern zu bekommen. - Er kann dabei die Bettenauslastung seines Krankenhauses aktualisieren. - Er kann auch die Gesamtanzahl an Betten in seinem Krankenhaus festlegen.
IT-Teams	Fremd	- Ihr Interesse liegt in der Wartung des Systems. - Sie haben Zugriff auf die technische Komponente und den Quellcode des Systems, um die Wartungsarbeit durchführen zu können.

II- Randbedingungen

1. Technisch

Randbedingungen	**Erläuterung**
Implementierung in objektorientierter Sprache.	Die Entwicklung der Software soll mit einer objektorientierten Sprache erfolgen.
Moderate Hardware-Ausstattung	Das System soll auf normal ausgestatteten Geräten laufen.
Entwicklungswerkzeuge	Eine von Jetbrains veröffentlichte IDE
Deployment	Als Docker Container

2. Organisatorisch

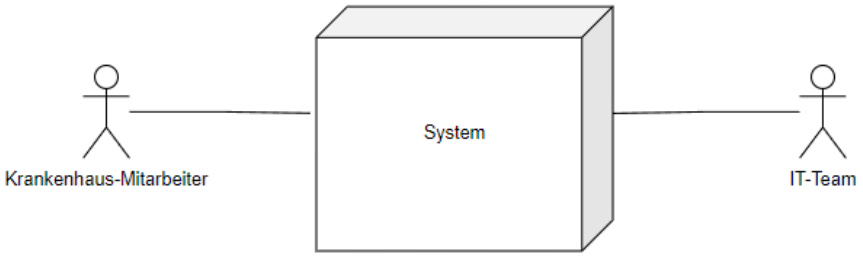
Randbedingungen	Erläuterung
Team	Entwickler: Kacem Mbarek, Kosai Alzaeim (Scrum-Master), Osamah Rafea, Dhruv Talaviya Begleiter: Hr. Becke, Hr. Matthiesen
Zeitplan	Beginn der Entwicklung ist Anfang WiSe23-24. Mindestens eine Online-Meeting pro Woche mit allen Teammitgliedern. (hier sind nur die Entwickler gemeint)
Vorgehensmodell	Iteratives Vorgehensmodell (Scrum)
Versionsverwaltung	Gitlab

3. Konventionen

Randbedingungen	Erläuterung
Dokumentation	Nach Arc42-Template
Kodier Richtlinien für gewählte Sprache	Die Coding Conventions der jeweiligen Sprache müssen eingehalten werden

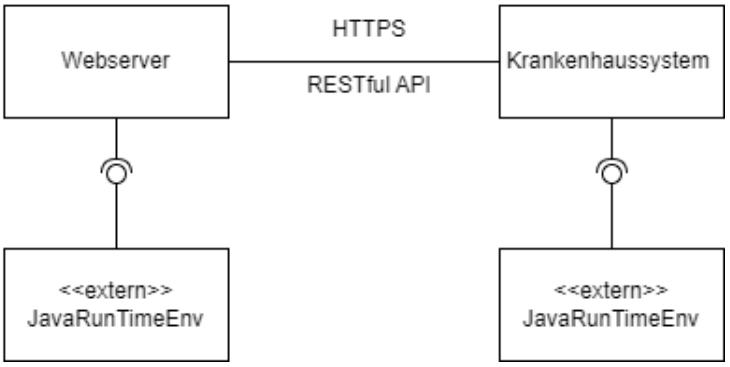
III- Kontextabgrenzung

1. Fachlicher Kontext:



Komponente	Beschreibung
Krankenhaus-Mitarbeiter	Auslastung abfragen und aktualisieren (in seinem Krankenhaus).
IT-Team	Wartung- und administrative-Aufgaben durchführen.

2. Technischer Kontext



Komponente	Beschreibung
Webserver	dient als Aggregator für die Daten, die aus dem System empfangen werden.
HTTPS	um Daten von Krankenhaus Systemen zu empfangen und dem Krankenhäuser-Observer bereitzustellen.
RESTFull API	Um Daten zwischen dem Webserver und den Krankenhaus Systemen auszutauschen

IV- Lösungsstrategie

1. Model-View-Controller (MVC):

➔ **Model:**

- ◆ Verantwortlich für die Geschäftslogik und Datenzugriffsoperationen.
- ◆ In Java implementiert, möglicherweise unter Verwendung von Spring Data JPA für den Datenzugriff auf H2 bzw. MySQL.

➔ **View:**

- ◆ Stellt die Benutzeroberfläche dar, die dem Benutzer angezeigt wird.

- ◆ Implementiert in JavaScript, möglicherweise mit Zusatzbibliotheken oder Frameworks wie React oder Angular.

➔ **Controller:**

- ◆ Dient als Schnittstelle zwischen Model und View.
- ◆ Nimmt Benutzeranfragen entgegen, leitet sie zur Verarbeitung an das Modell weiter und sendet die Antwort zurück an die View.

2. Einsatz von Spring Boot und Spring Security:

- ➢ Vereinfacht die Konfiguration und den Start des Projekts.
- ➢ Bietet eine eingebettete Tomcat-Instanz, was das Deployment und Testing vereinfacht.
- ➢ Ermöglicht die einfache Integration von Spring Security und anderen benötigten Spring-Komponenten.

3. Frontend-Entwicklung mit JavaScript:

- ➢ Verantwortlich für die Erstellung dynamischer und interaktiver Benutzeroberflächen.
- ➢ Kann mit weiteren Bibliotheken oder Frameworks wie React.js, Angular erweitert werden

4. Skalierung:

Jeder Knoten speichert die Informationen über andere bekannte Knoten in einer Liste . Wenn ein neuer Knoten dem Netzwerk hinzugefügt werden soll, etabliert dieser neue Knoten eine Verbindung zu einem beliebigen bereits bekannten Knoten im Netzwerk. Über Remote Procedure Calls (RPC) sendet dieser bereits existierende Knoten dem neuen Knoten seine Liste bekannter Knoten. Der neue Knoten richtet daraufhin eine Verbindung zu jedem Knoten in dieser Liste ein.

5. Middleware-Einsatz

Funktion der Middleware:

- **Datenaustausch:** Ermöglicht das Senden und Empfangen von Daten zwischen dem zentralen Server und den Krankenhaus-Knoten. Dies beinhaltet Bettenauslastung und administrative Daten.

Remote Procedure Call (RPC):

- Die Middleware wird durch ein Remote Procedure Call (RPC) Framework implementiert. Dieses Framework ermöglicht es, Methoden oder Prozeduren auf entfernten Knoten aufzurufen, als ob sie lokal wären.

6. Thread-Management:

- Threads werden eingesetzt, um parallele Verbindungen und Kommunikationsprozesse zu handhaben. Jeder Knoten kann einen Thread haben, der auf eingehende Nachrichten hört, und separate Threads für das Senden von Nachrichten.

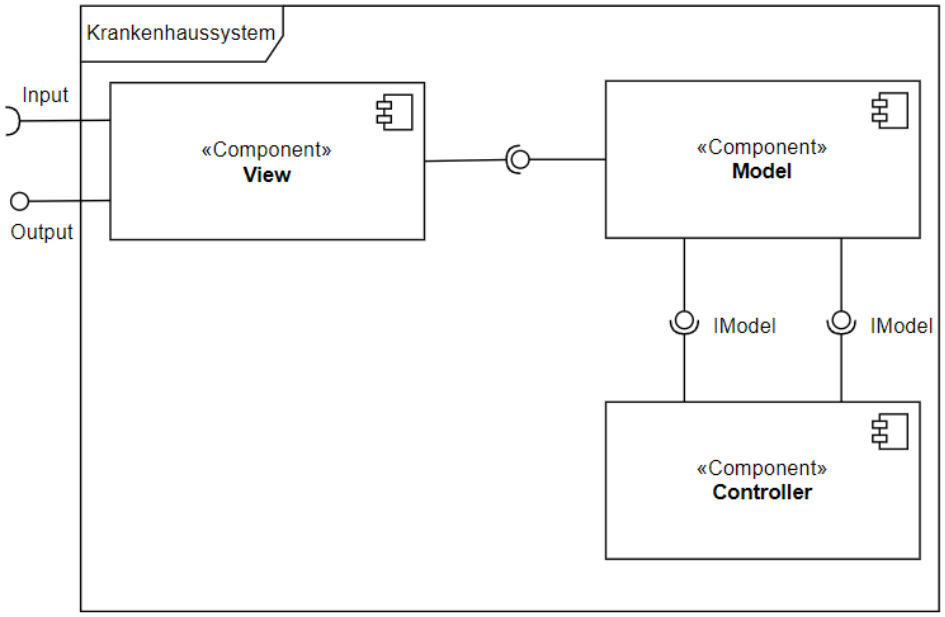
7. Bully-Algorithmus Implementierung:

- Kriterien für die Leader-Wahl werden festgelegt (z.B. **Knoten-ID,** Rechenleistung).
- Eine Logik wird implementiert, die einen Wahlprozess initiiert, wenn der aktuelle Leader ausfällt oder ein neuer Knoten dem Netzwerk beitritt.

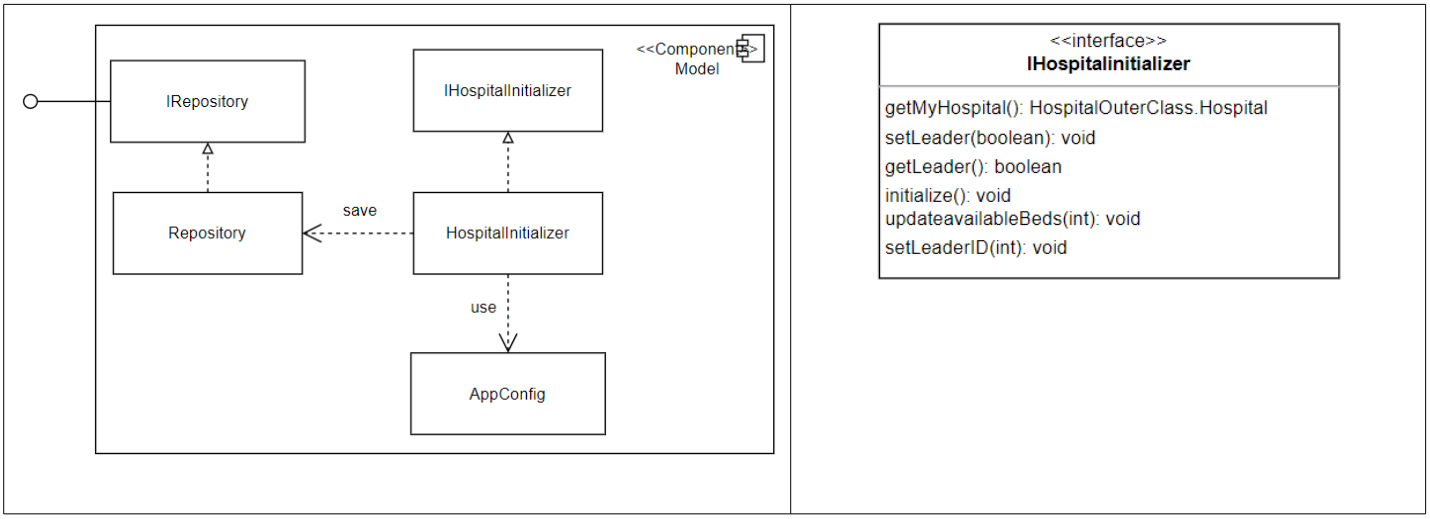
8. Modulares Design:

- Der Code wird in logischen Modulen organisiert, z.B. ein Modul für Netzwerkkommunikation, eines für den Bully-Algorithmus, usw.
- Dies erleichtert die Wartung und das Testen des Systems.

V- Bausteinsicht:



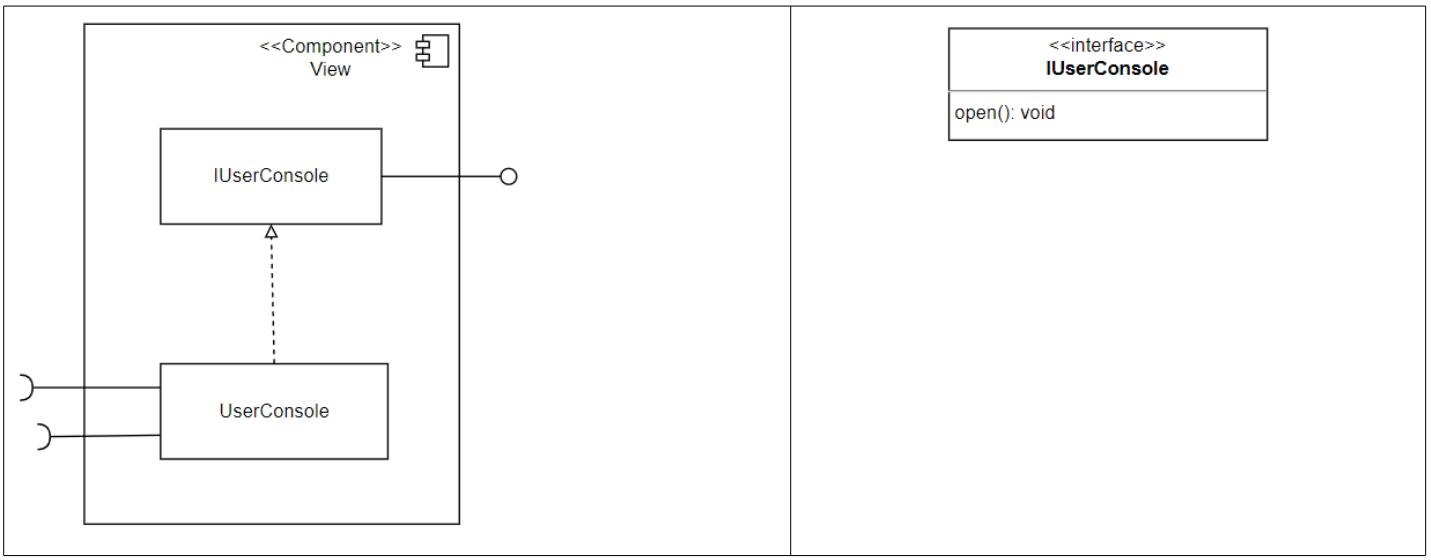
Komponente	Beschreibung
view	die Komponente, die für die Darstellung der Benutzer-Nachrichten und Meldungen.
Model	repräsentiert das Modell (Krankenhaus)
Controller	beinhaltet die Implementierung-Logik



Komponente	Beschreibung
Repository	Beinhaltet alle Krankenhäuser im System.
HospitalInitializer	baut ein Krankenhausobjekt.
AppConfig	Einstellungsdatei, um ein Krankenhausobjekt zu bauen

Methode	Beschreibung
getMyHospital	gibt das Krankenhausobjekt, in dem aktuellen Knoten zurück.
setLeader	stellt eine Krankenhaus-Knote als Leader ein/ab.
initialize	benutzt die AppConfig, um ein Krankenhausobjekt zu inizializieren.
updateAvailableBeds	aktualisiert die verfügbaren Betten in einem Krankenhaus.

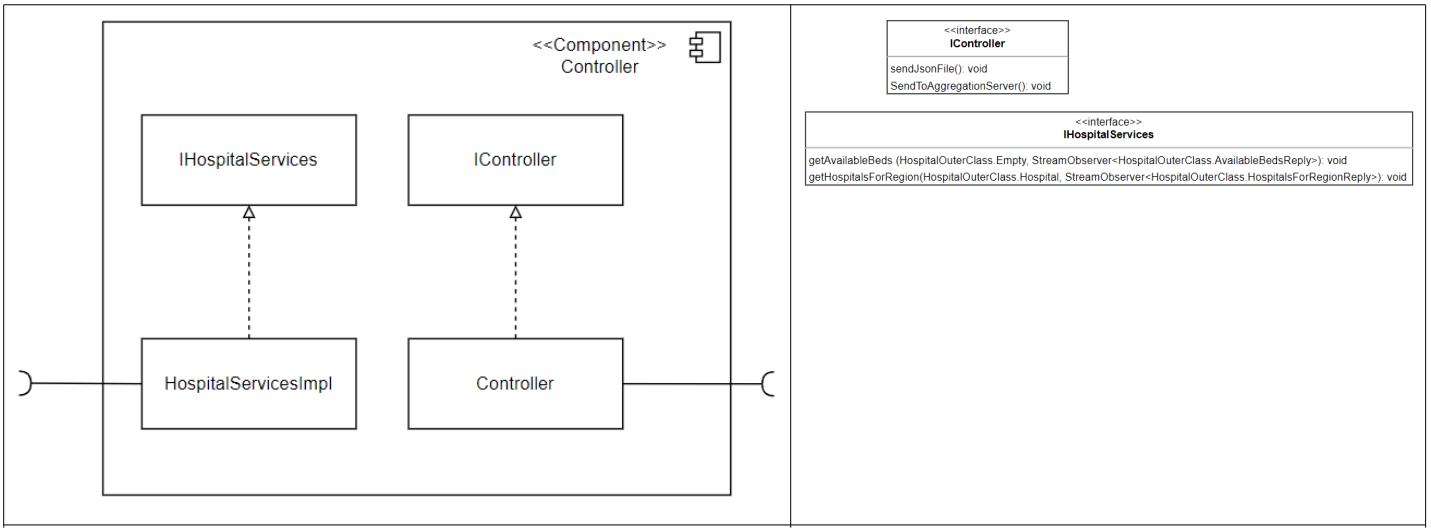
setLeaderID	speichert die ID vom Leader im System.



Komponente	Beschreibung
UserConsole	für die Kommunikation zwischen dem Benutzerund dem System.

Methode	Beschreibung
open	Die Methode nimmt die Eingabe von Benutzer auf und liefert die gewünschten Daten abhängig von der Benutzereingabe. Eingabeliste:

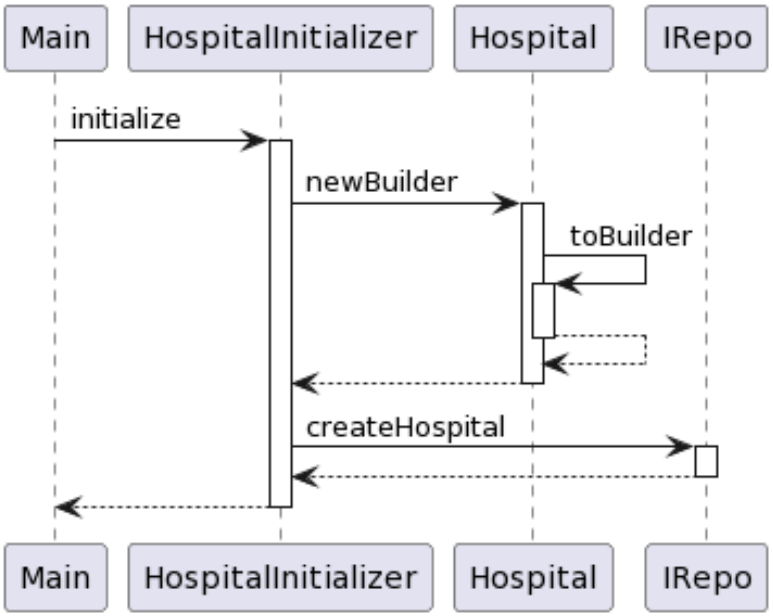
	– setAB: um die Anzahl an freien Betten zu ändern. – getHI: liefert alle Informationen über das Krankenhaus. – getAHI: liefert alle Informationen über alle Krankenhäuser im System. – whoIsLeader: liefert die ID des Leaders im System. – clear: löscht die (alten) Nachrichten von der Console – help: um die verschiedenen möglichen Eingaben zu zeigen.



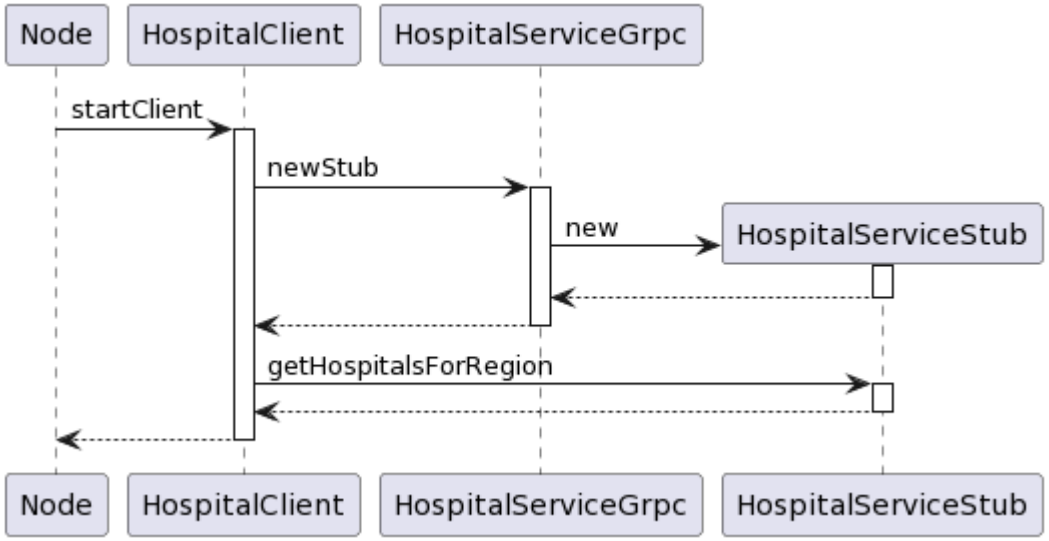
Komponente	Beschreibung
Controller	die vom Leader empfangenen Daten zum Server schicken.
HospitalServiceImpl	extrahiert die Informationen aus den Krankenhausobjekte.

Methode	Beschreibung
sendJsonFile	Sendet eine JSON-Datei an einen Server über HTTP-POST.
sendToAggregationServer	Sendet gesammelte Krankenhausinformationen als JSON-Datei an einen Aggregations-Server.
getavailableBeds	Sendet die Anzahl der verfügbaren Betten des aktuellen Krankenhauses.
getHospitalForRegion	Verarbeitet ein Krankenhausobjekt, fügt es dem Repository hinzu und sendet als Antwort eine Liste aller Nachbarn.

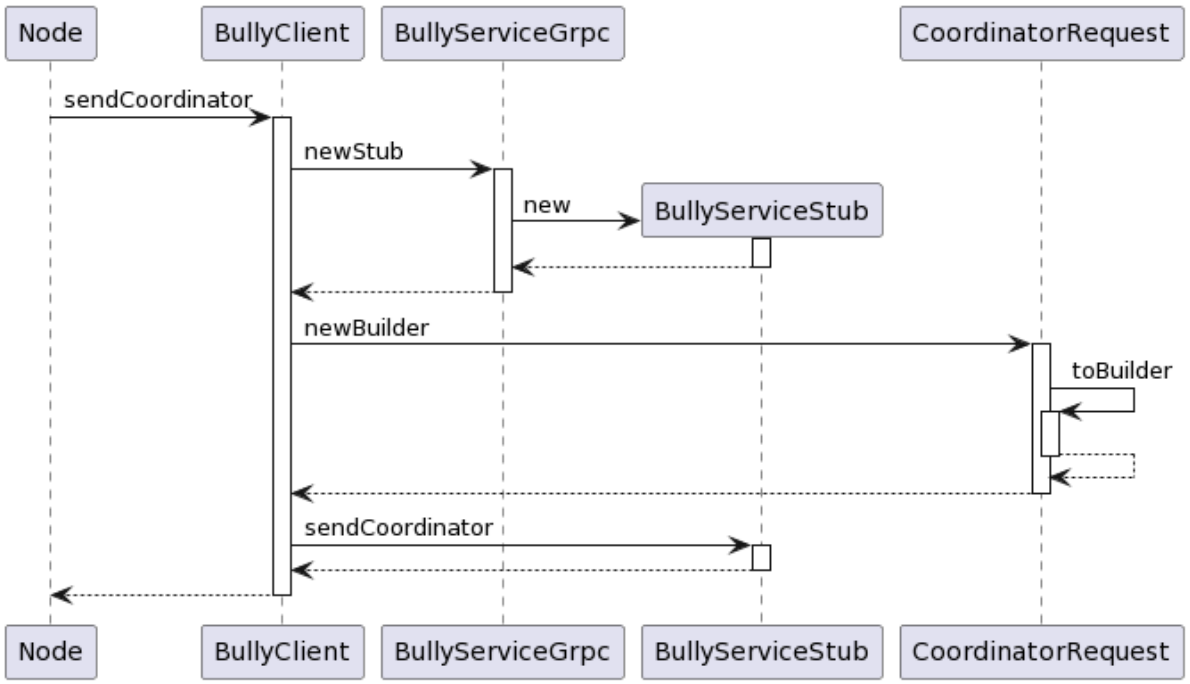
VI- Laufzeitsicht:



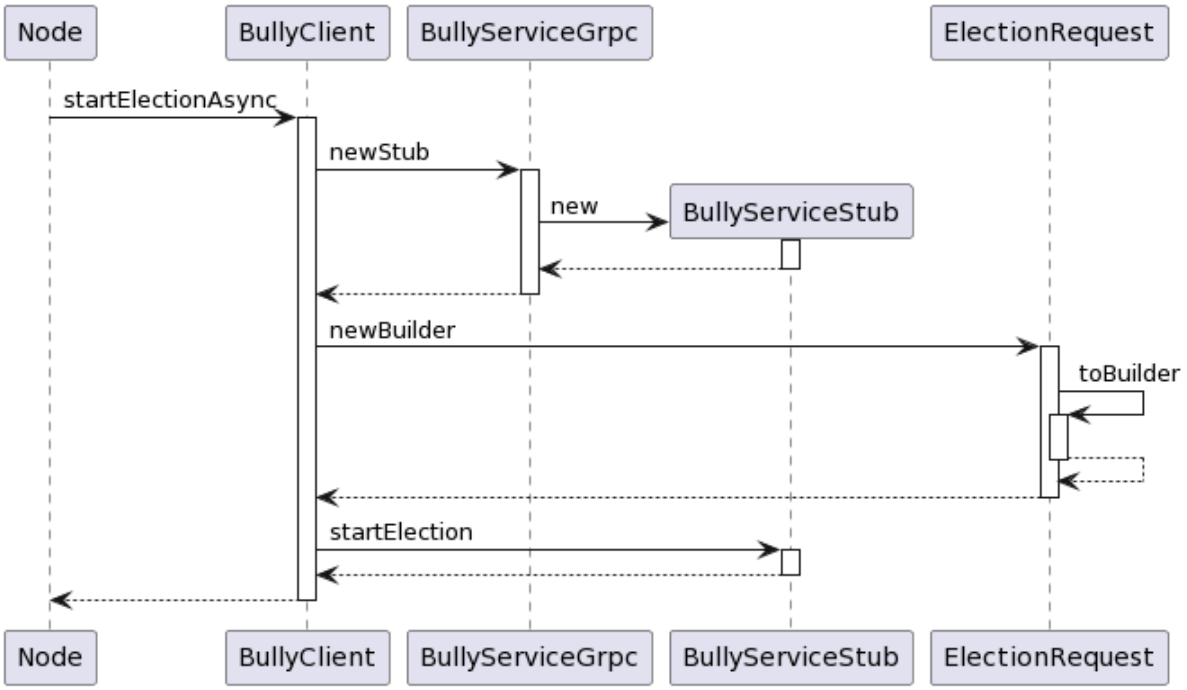
inizialize(): liest die Config-Datei, erzeugt ein Krankenhaus Objekt und speichert es im Repository.



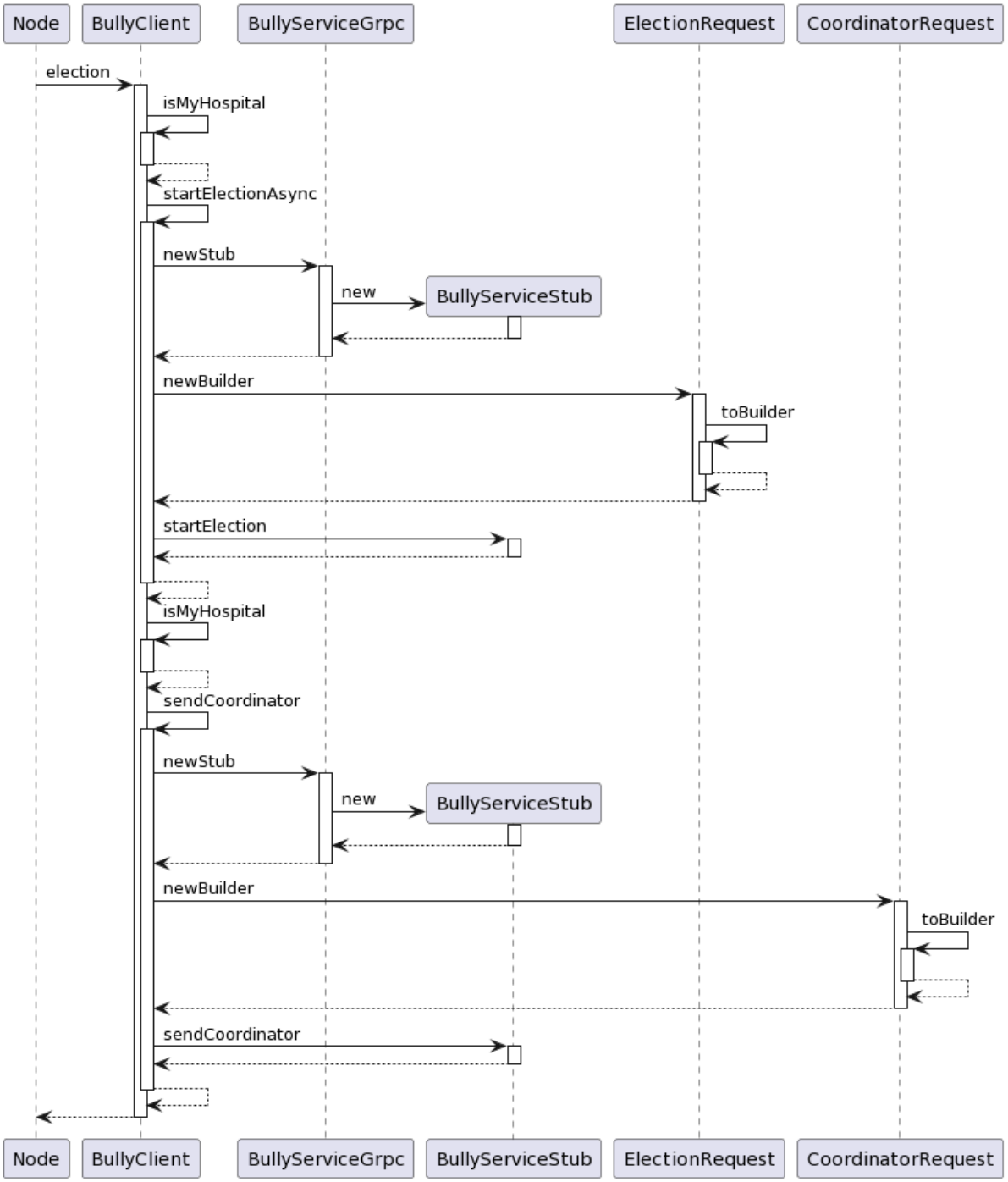
startClient(): erzeugt einen Client-Stub und ruf auf die anderen Knoten die Methode getHospitalforRegion()



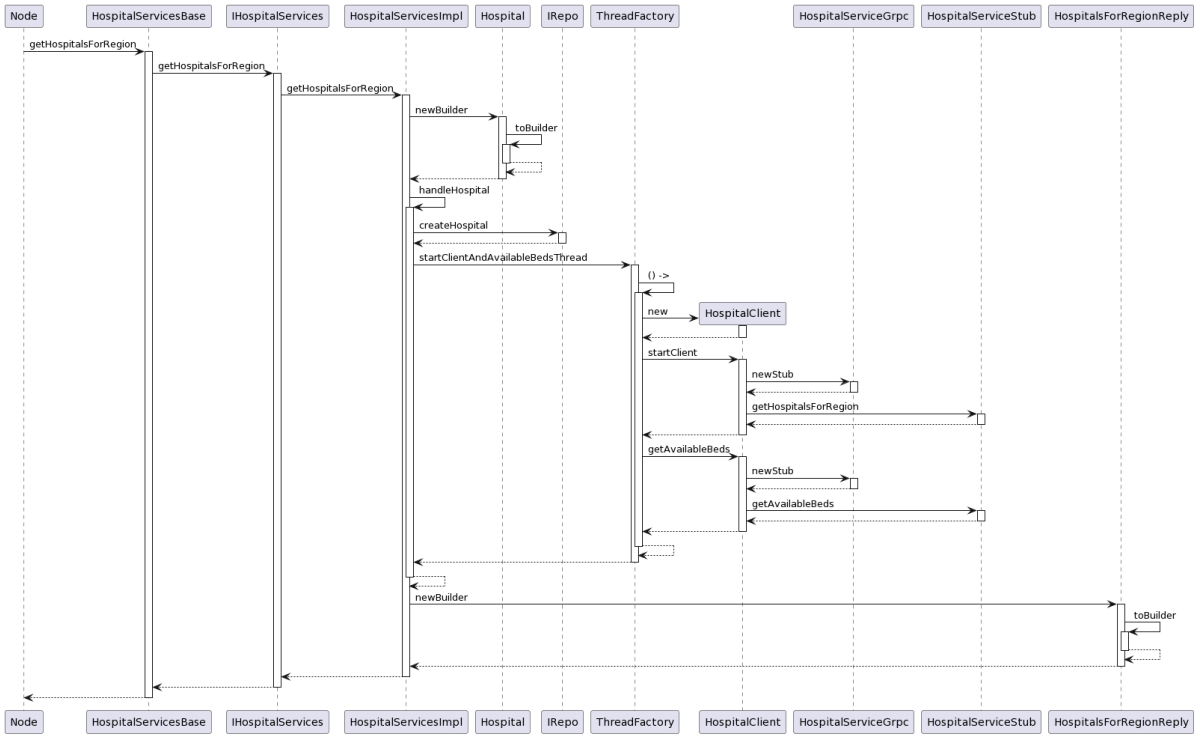
sendCoordinator(): erstellt einen Bully-Client-Stub, und schickt allen anderen Knoten eine Coordinator-Response wann er Coordinator gewählt wird.



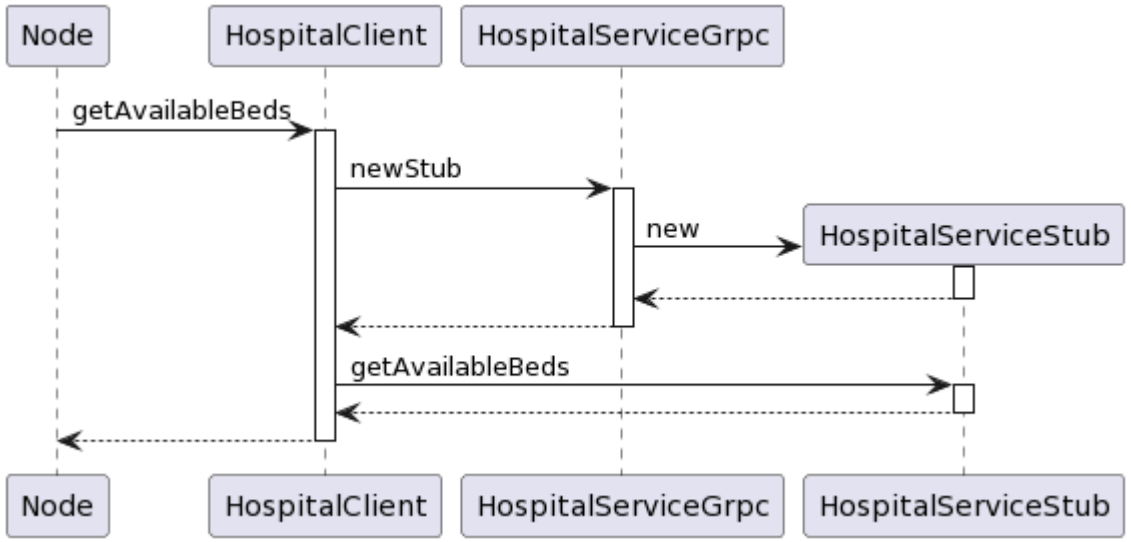
sendElectionAsync(): starte eine Election-Anfrage auf den größeren Knoten.



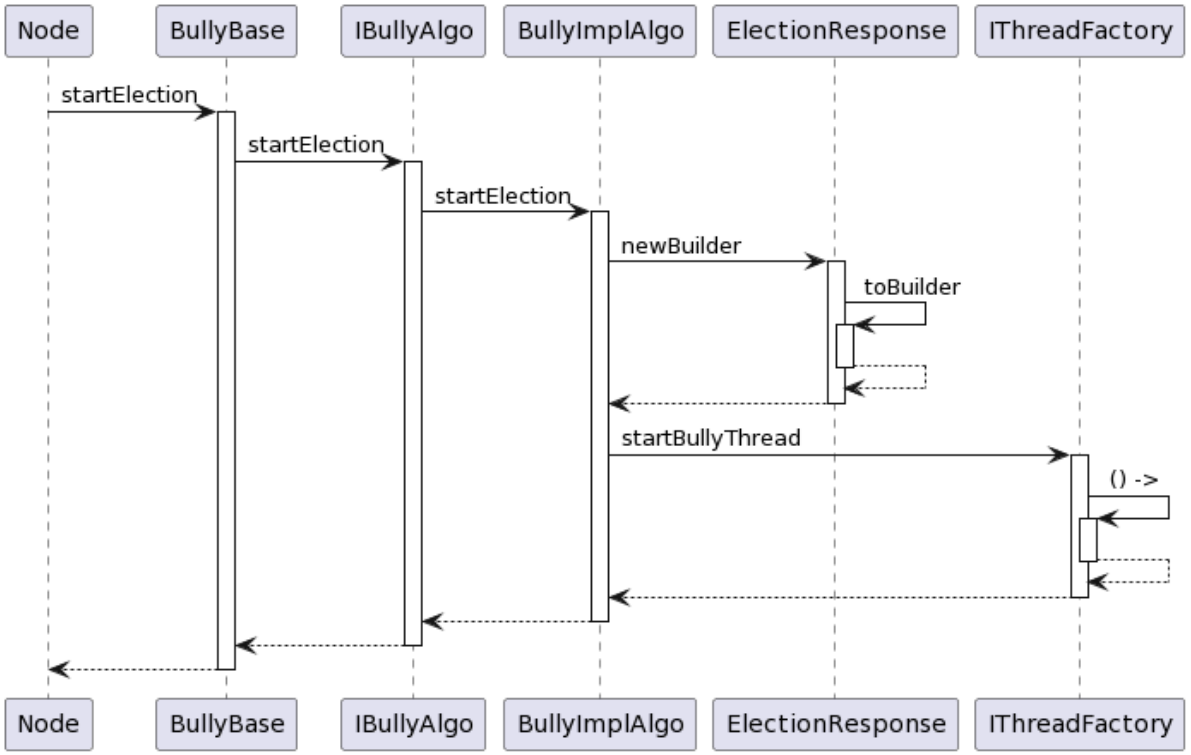
election(): die Methode wird aufgerufen, wenn ein neuer Knoten hinzugefügt wird oder wenn der Leader ausfällt.



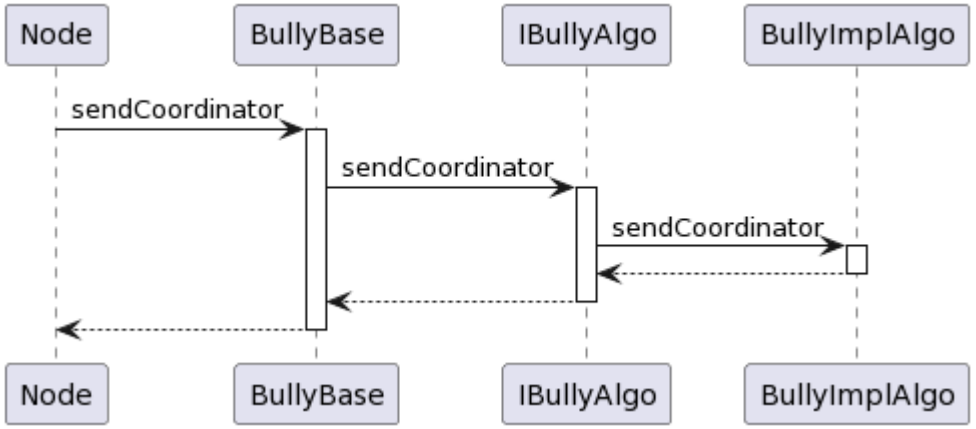
getsHospitalForRegion(): liefert den anderen Knoten alle Informationen von den Nachbarknoten und fragt alle 3 Sekunden nach Änderung an den freien Betten.



hospitalClientGetavailableBeds(): erzeugt einen Client-Stub und ruft (invoke) getAvailableBed() auf die Nachbarknoten.

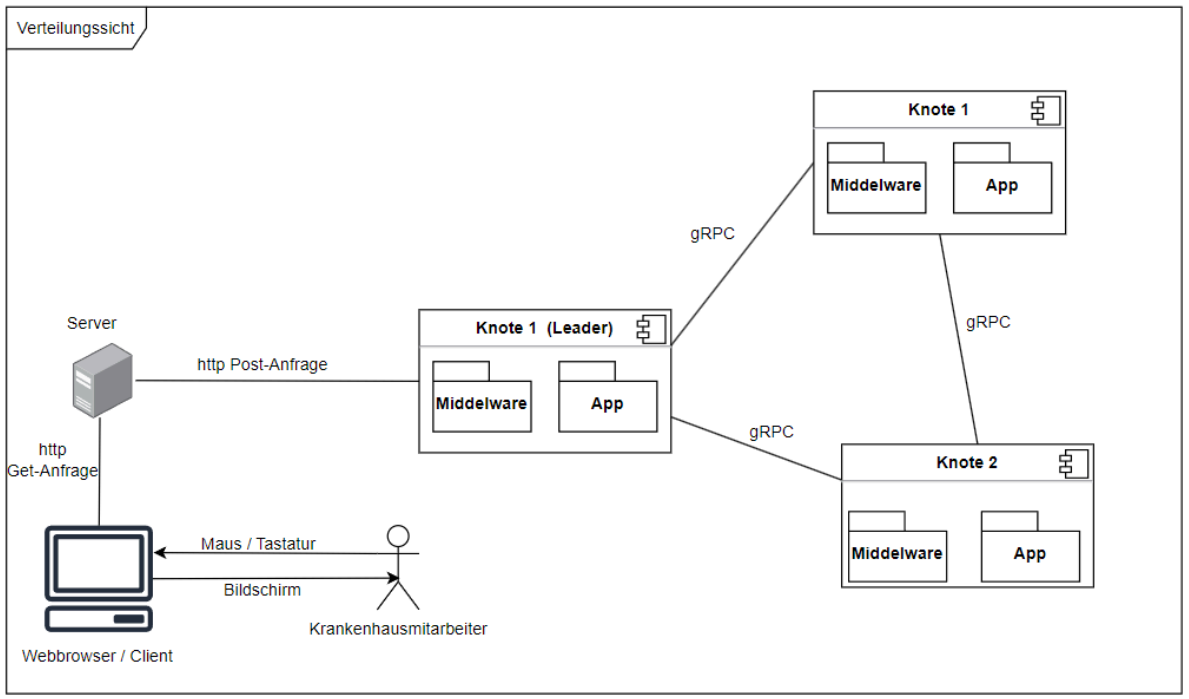


startelection(): eine Election wurde gestartet. Wenn eine Antwort als True empfangen wird, wird der Election-Prozess bei den Knoten weiter gemacht, die einen größeren ID haben.



sendCoordinator(): alle anderen Knoten werden vom neuen Leader benachrichtigt.

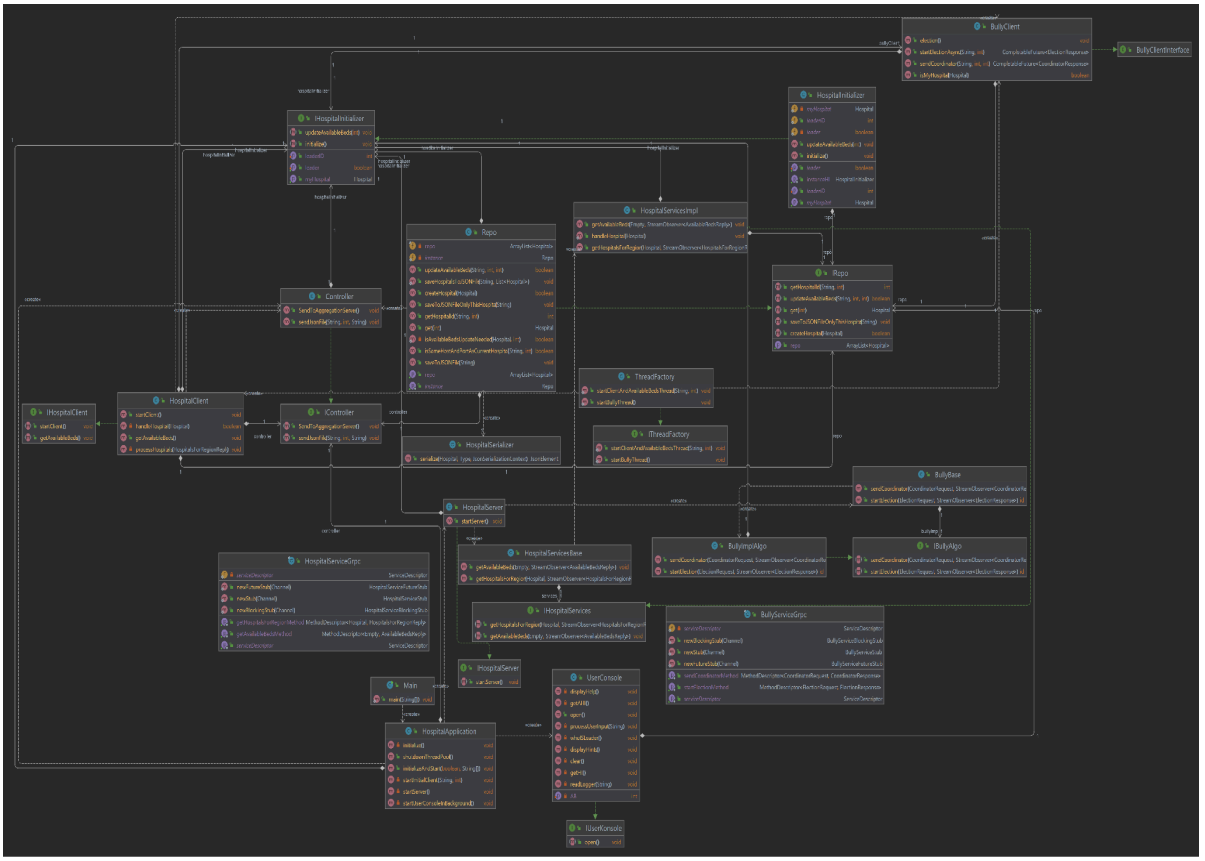
VII- Verteilungssicht



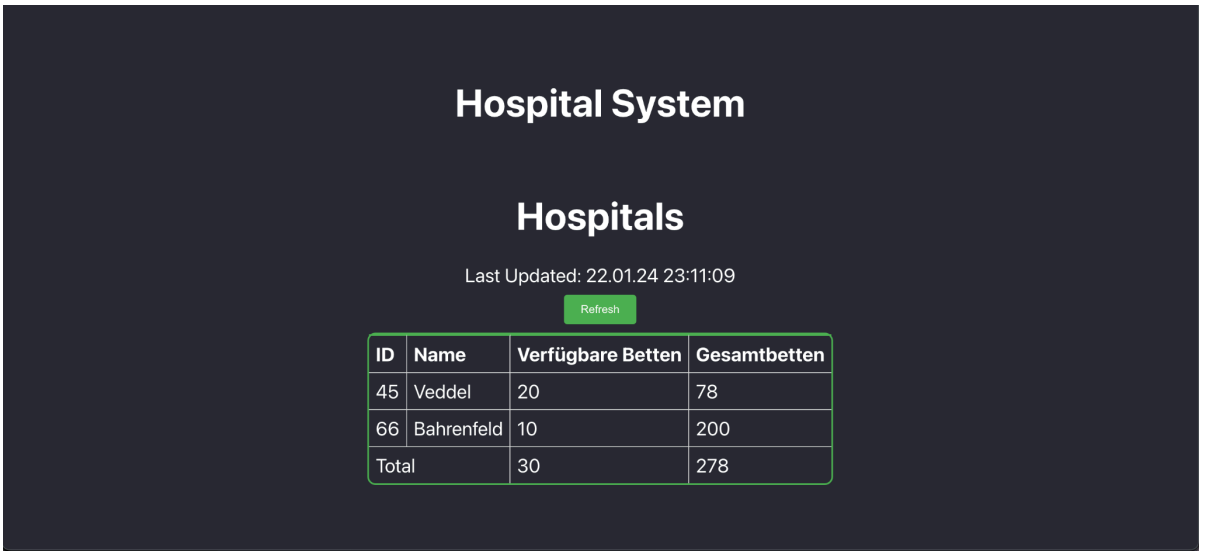
- Alle Knoten sind zusammen verbunden. Sie kommunizieren mit RPC, um Daten auszutauschen und einen Leader zu wählen.
- Alle Knoten tauschen sich alle 3 Sekunden ihre eigene Anzahl an freien Betten. Somit hat jedes Krankenhaus die gesamten Überblick über das ganze Netzwerk.
- Der Leader hat die Aufgabe, seine gesamte Übersicht zum Server zu schicken, sobald es eine Änderung gibt.
- Ein Benutzer benutzt ein Endgerät, um die Auslastung im Krankenhaus- Netzwerk zu kontrollieren.
- Zum Server wird eine Get-Anfrage geschickt, auf der der Server mit einer Post-Anfrage antwortet, die die gewünschten Daten enthält.

VIII- QUERSCHNITTLICHE KONZEPTE

1. Klassendiagramm:



2. Benutzeroberfläche:



Auf der Benutzeroberfläche kann man eine detaillierte Übersicht über die Krankenhäuser sehen. Man kann

leicht die verfügbaren Betten in jedem Krankenhaus ablesen. Dabei kann man auch von dem Zeitstempel ablesen, wann die letzte Aktualisierung war.

IX- Entscheidungen:

1. RPC Lösung:

Wir haben uns für gRPC als Kommunikationsprotokoll entschieden, da es automatische Codegenerierung für Clients und Server ermöglicht. Diese Automatisierung erleichtert die Entwicklung und Wartung erheblich, insbesondere beim Marshalling und Unmarshalling von Daten zwischen Client und Server. Darüber hinaus unterstützt gRPC bidirektionale Streams, was bedeutet, dass sowohl der Client als auch der Server gleichzeitig Daten senden können. Dies ist besonders vorteilhaft für Echtzeitkommunikation. Die Entscheidung für gRPC basiert auf der Effizienz, Skalierbarkeit

und Flexibilität, die es für unser verteiltes System bietet. Die bidirektionale Kommunikation und die Codegenerierung tragen maßgeblich zu diesen Vorteilen bei. Beachtenswert ist jedoch, dass gRPC im Vergleich zu XML-RPC und JSON-RPC als etwas komplexer wahrgenommen werden kann, und das Debuggen möglicherweise als anspruchsvoller empfunden wird.

2. **Singleton-Designmusters:**

   Wir haben uns für Singleton-Designmusters entschieden, da wir an verschiedenen Stellen mit einer Instanz von einem Objekt arbeiten sollen. Singleton-Designmusters ermöglicht uns, eine einzige Instanz von z.B einem Krankenhausobjekt in einem Knoten zu haben und mit der weiter zu arbeiten. und so wird in jedem Knoten eine Instanz verwendet.