Start: 28 Oktober 2023

Version 1.4

Einführung und Ziele

Es wird im Verlauf des Praktikums eine Visualisierung eines Konsens Algorithmus, welcher auf Flooding basiert, dabei wird der Algorithmus mithilfe von Containern realisiert.

Um eine bessere Vorstellung zu bekommen, wird ein Anwendungsbeispiel gewählt, wo jeder Knoten eine Aufgabe mit einer Priorisierung bekommt. Um es genauer zu beschreiben, gibt der “Chef” seinen Mechatronikern eine Liste von Autos, die repariert werden müssen. Die “Mechatroniker”-Knoten entscheiden je nach Dringlichkeit des Autos, das die höchste Priorität hat, welches Sie zuerst angehen und reparieren.

Aufgabenstellung

Kürzel	Anforderung	Erläuterung
A1	Konsens Algorithmus basierend auf Flooding	Ein Konsensalgorithmus hat das Ziel, in einem verteilten System eine gemeinsame Entscheidung oder einen Konsens über einen Wert oder Zustand zu erreichen.
A2	Messung der Leistungsfähigkeit	Es sollen Messungen durchgeführt werden, um die Leistungsfähigkeit zu zeigen. Methodisch kann man sich an der Messung von Algorithmen in AD orientieren.
A3	Visualisierung sollte skalierbar sein	Das heißt, es sollte mit N verschiedenen Knoten/Containern funktionieren.
A4	Container Lösung	Keine Simulation des Algorithmus, sondern eine Realisierung des mit Hilfe einer Container-Lösung.
A5	Flooding Algorithmus	Der Flooding-Algorithmus flutet das Netz mit Nachrichten. Flooding Pseudo Code:

		function flooding(node, message):
		message.markAsVisited()
		
		for neighbor in node.getNeighbors():
		if not message.hasVisited(neighbor):
		neighbor.receiveMessage(message)
		flooding(neighbor, message)
		
		class Message:
		visitedNodes = set()
		
		def markAsVisited():
		visitedNodes.add(self.sender)
		
		def hasVisited(node):
		return node in visitedNodes
		
		class Node:
		neighbors = []
		
		def getNeighbors():
		return neighbors
		
		def receiveMessage(message):
		processMessage(message)

A6	Priorisierung von Aufgaben in einem verteilten System	Jede Aufgabe hat eine Priorität, wie dringend sie bearbeitet werden muss. Es ist wichtig, dass alle Server im System wissen, welche Aufgabe die höchste Priorität hat, damit sie entsprechend reagieren können.

Qualitätsziele

Qualitätsziel	**Erläuterung**
Funktionale Eignung	Das System sollte den Anforderungen des Kunden entsprechen und nicht davon abweichen. Das System wird entwickelt, um eine Vielzahl von funktionalen Anforderungen zu erfüllen, die in enger Zusammenarbeit mit den Stakeholdern definiert werden. Die Anforderungen erstrecken sich über verschiedene Bereiche, von grundlegenden Benutzerinteraktionen bis hin zu komplexen Systemprozessen.
Zuverlässigkeit	Das verteilte System läuft zuverlässig und eventuelle Fehlerzustände sind

	nachvollziehbar. Die Zuverlässigkeit unserer Software ist von zentraler Bedeutung, da sie direkten Einfluss auf die Stabilität, Verfügbarkeit und Leistungsfähigkeit des Systems hat. In diesem Abschnitt wird die Verlässlichkeit der Software hinsichtlich verschiedener Aspekte näher betrachtet, um sicherzustellen, dass sie den Anforderungen unserer Benutzer und Stakeholder gerecht wird.

Stakeholder

Rolle	Kontakt	Erwartungshaltung	Anforderung
Kunde	Martin Becke, Frank Matthiesen	Die Anforderungen sollen der Referenzliteratur entsprechen. Der Code ist durch umfassende Tests abgesichert. Die Dokumentation wurde entsprechend gestaltet und die Deadline für die Prüfungsleistung (PVL) wird am Ende des Semesters erreicht.	A1-A5
Entwickler	Hüseyin Akkiran, Onur Aslan	- PVL erhalten - Lernerfolg mit verteilten Systemen	A6

Randbedingungen

Technische Randbedingungen

Randbedingung	Erläuterung
Java (JRE)	Java ist die am meisten verwendete objektorientierte Programmiersprache des Teams, weshalb es in der Programmierung leichter ist. Java 17 wird verwendet.
IntelliJ IDEA Entwicklungsumgebung	IntelliJ IDEA ist eine integrierte Entwicklungsumgebung (IDE) des Softwareunternehmens JetBrains für die Programmiersprachen Java, Kotlin, Groovy und Scala. Außerdem bietet uns IntelliJ Features wie Apache Maven, JUnit, einen GUI-Editor, Tools zur Versionskontrolle, insbesondere Git, sowie hauptsächlich verschiedene Möglichkeiten zum

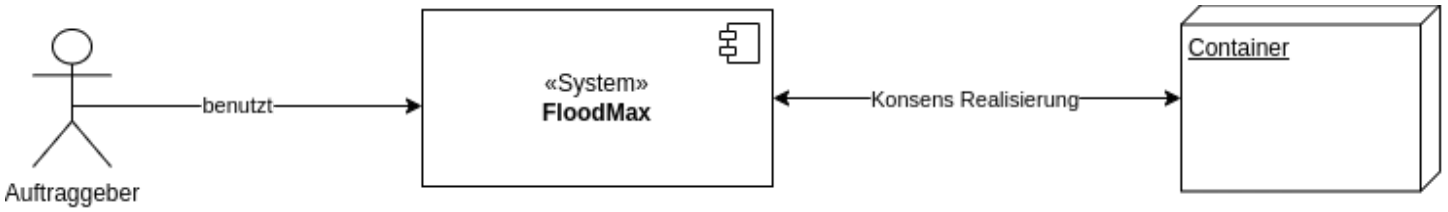
	automatischen Refactoring von Code.
Docker Container	Docker Container dienen zur Virtualisierung auf Anwendungsebene, bei der Anwendungen und alle benötigten Ressourcen in einem isolierten Paket verpackt werden.

Organisatorische Randbedingungen

Randbedingung	**Erläuterung**
Team	Onur Aslan und Hüseyin Akkiran arbeiten eng zusammen, teilen sich die Aufgaben gerecht auf und nutzen sowohl Pair-Programming als auch individuelles Programmieren, um effizient zu arbeiten.
Vorgehensmodell	Ein agiler Ansatz wird hier gewählt, es ist für das Team am einfachsten und somit kann schon neben der Dokumentation eine Code Basis entstehen.
Git (Gitlab)	Git und Gitlab ermöglichen eine effiziente, kollaborative Projektarbeit durch synchronisierte Zusammenarbeit und die Möglichkeit, in separaten Branches zu arbeiten.
Testprozess	JUnit spielt eine entscheidende Rolle in unserem Testprozess, wo wir sicherstellen, dass das System korrekt funktioniert. Während wir nicht immer Test-Driven Development (TDD) anwenden, nutzen wir dennoch JUnit, um in verschiedenen Phasen der Entwicklung die Qualität der Anwendung zu gewährleisten.
Zeitplan	Start des Projekt war der 28.10.2023, der erste Prototyp und eine grundfunktionalität der Applikation sollte bis 20.12.2023 erreicht sein und über die kommenden Weihnachtsferien sollte am Ende des Jahres 2023 diese Dokumentation so gut wie fertig sein und meistens nur noch programmiert werden.
Abhängigkeitsmanagement	Maven wird benutzt, um Bibliotheken einzufügen und zu verwalten.

Kontextabgrenzung

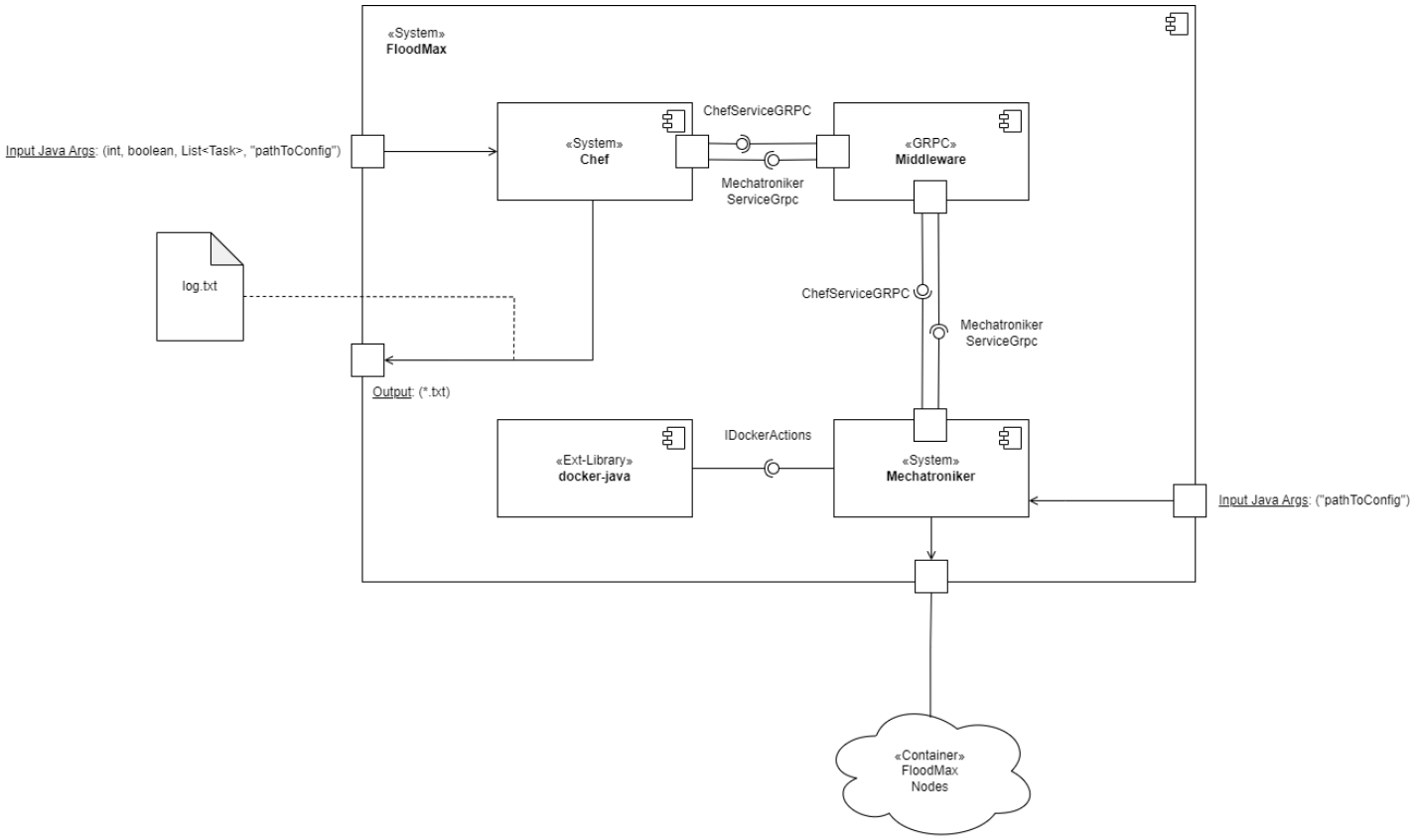
Fachlicher Kontext



Nachbarsystem/Benutzer	Beschreibung	Input	Output
Auftraggeber	Benutzer des FloodMax Systems können eine Liste von Aufgaben übermitteln, die mit dem Konsens Algorithmus bearbeitet werden. Dabei kann er auch angeben, wie viele Knoten verwendet werden sollen.	Eine Auftragsliste mit Aufgaben, welche mittels des Konsens Algorithmus gelöst werden sollen. Sowie auch die Anzahl an Knoten die benutzt werden sollen um das Problem zu lösen	Visuelle Ausgabe auf einer Konsole, wo der Status angezeigt wird. (optional) Logfile, eine optionale Ausgabe, die an den Benutzer bereitgestellt werden kann für weitere Informationen
Container	Container, speziell Docker Container, als Knoten für die Algorithmus Implementierung, Sie werden untereinander den Konsens durchführen.	Sockets	Sockets

Technischer Kontext

Task
- TaskName: String - Priority: int - Timestamp: timestamp
+ getTaskName(): String + getPrio(): int + getTime(): timestamp + toString(): String



log.txt = Logdatei für Ereignisse, die im Programm passieren.

"pathToConfig" = Ist eine .config Datei welche 3 Zeilen hat,

1- eigener Port den man anbietet

2- port der vom gegenüber angeboten wird

3- IP Adresse vom gegenüber

Lösungsstrategie

Verweis auf JavaDoc im Gitlab.

Anwendungsfälle

Nummer	UC-1
Titel	Eine Auto Aufgabenliste wird ausgeteilt
Akteur	Benutzer
Ziel	Lösen der ganzen Aufgabenliste
Auslöser	Aufgabenliste wird dem Chef übermittelt
Vorbedingung	1. Es gibt Aufgaben mit Priorität
Nachbedingung	Chef hat die Aufgaben angenommen.
Erfolgsszenario	1. Der Benutzer teil dem System die Aufgaben die er hat mit 2. Das System prüft die Aufgaben auf Richtigkeit 3. Das System arbeitet die Aufgaben ab a. Währenddessen teil das System dem Benutzer den aktuellen Stand mit 4. Sobald das System fertig ist, wird dem Nutzer Bescheid gegeben.
Fehlerfälle	Eine Aufgabe enthält keine Priorität

Nummer	UC-2
Titel	Mechatroniker ermitteln das Auto welches repariert werden soll
Akteur	System
Ziel	Die Aufgabe mit der höchsten Priorität erledigen
Auslöser	Mechatroniker möchte neue Aufgabe erledigen
Vorbedingung	Mechatroniker hat keine Aufgabe oder hat eine Aufgabe gelöst
Nachbedingung	Mechatroniker bekommt neue Aufgabe
Erfolgsszenario	1. Der Mechatroniker hat keine Aufgabe oder eine vorhandene Aufgabe erledigt.

	2. Das System geht die Aufgabenliste durch und prüft, welche Aufgabe die höchste Priorität hat. a. Es gibt mehrere Aufgaben mit gleicher Priorität 3. Das System gibt dem Benutzer die Aufgabe die die höchste Priorität hat und teilt sie dem Mechatroniker mit
Fehlerfälle	Es gibt mehrere Aufgaben mit gleicher Priorität

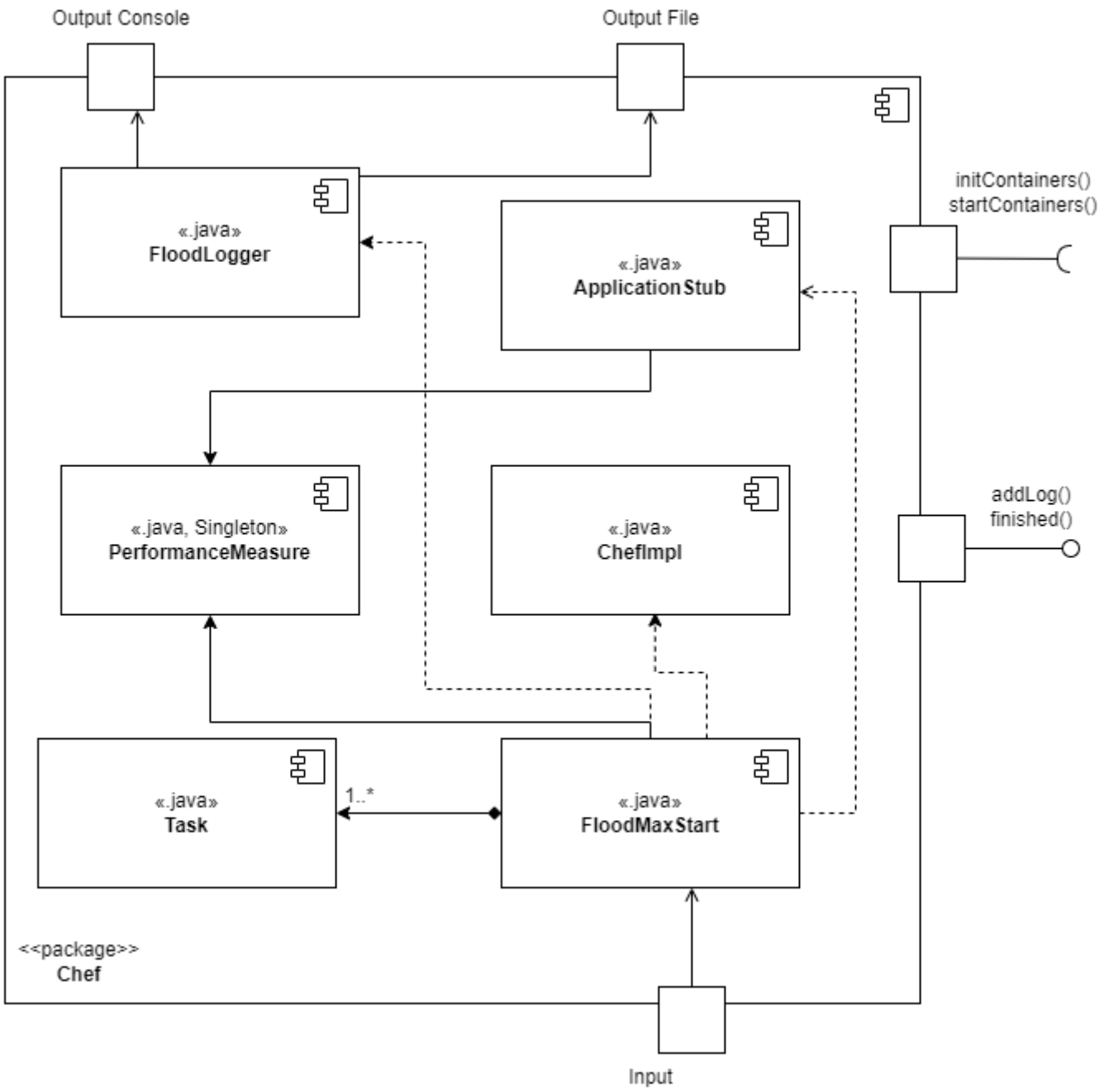
Nummer	UC-3
Titel	Mechatroniker melden aktuelle Arbeitsstände
Akteur	System
Ziel	Den Stand der Mechatroniker zu erfahren.
Auslöser	Mechatroniker haben einen Stand beendet.
Vorbedingung	Mechatroniker haben einen Stand beendet.
Nachbedingung	(Aktuelle Arbeitsstände der Mechatroniker wurden gemeldet)
Erfolgsszenario	Mechatroniker hat seinen Arbeitsstand abgesendet. 1. Arbeitsstand wird vom Mechatroniker bearbeitet 2. Ein Arbeitsstand wird abgeschlossen 3. Nachricht wird gesendet das dieser Arbeitstand abgeschlossen ist a. Mechatroniker wartet nicht und macht mit der Arbeit weiter
Fehlerfälle	Mechatroniker sendet gar keine Stände.

Nummer	UC-4
Titel	Chef ermittelt die Zeit für die Abarbeitung eines Autos.
Akteur	System
Ziel	Akkurate Messung der genutzten Zeit
Auslöser	Die Mechatroniker machen sich an die Aufgaben ran.
Vorbedingung	Es darf kein Mechatroniker arbeiten
Nachbedingung	Alle Mechatroniker haben ihre Arbeit abgelegt und sind fertig.
Erfolgsszenario	1. Chef wartet bis die Mechatroniker Anfangen

	2. Sobald Mechatroniker anfangen zu arbeiten wird die Zeit gestartet a. Ermittle ebenfalls die Zeit der einzelnen Aufgaben 3. Sobald alle Mechatroniker die Arbeit abgelegt haben und wenn keine Aufgaben mehr vorhanden sind, stopp die Zeit.
Fehlerfälle	Obwohl alle Aufgaben bearbeitet worden sind, legt ein Mechatroniker die Arbeit nicht ab.

Bausteinsicht

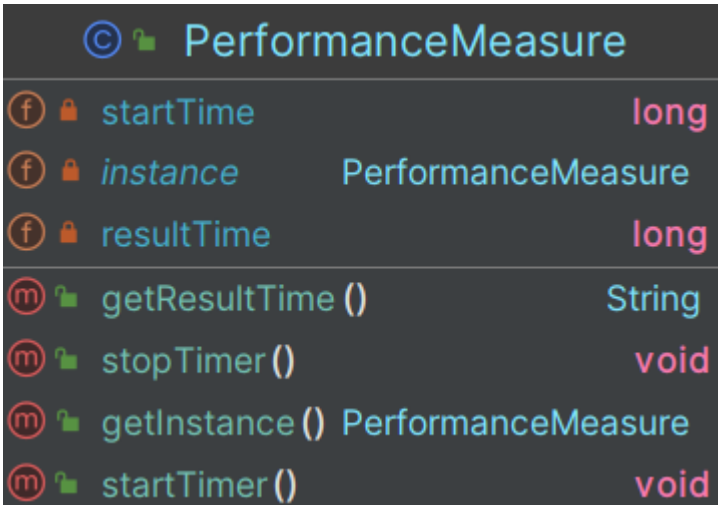
Ebene 1

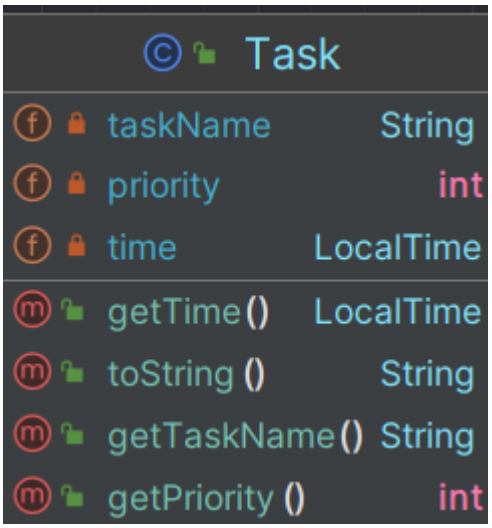


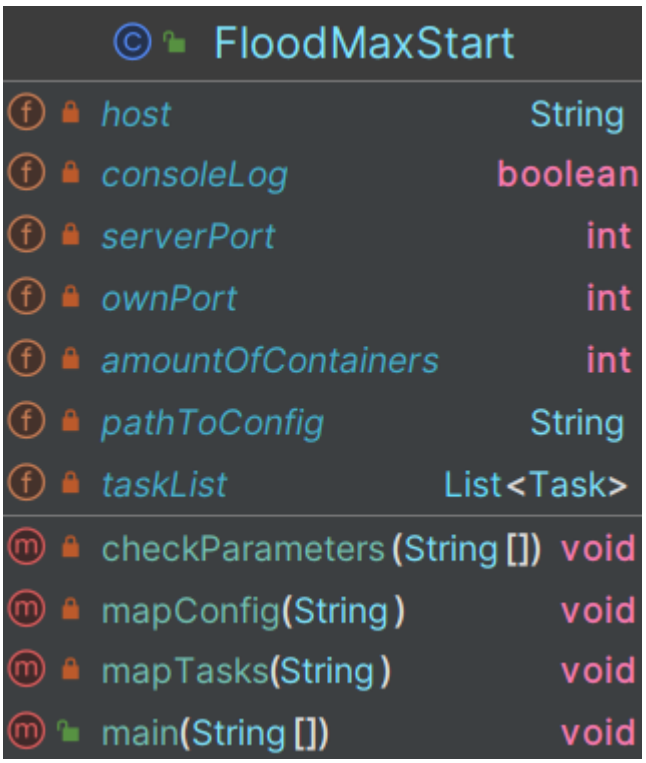
Block	**Beschreibung**
FloodMaxStart	Start des Systems, die Eingaben von dem Benutzer werden hier verarbeitet. Start des Algorithmus. Flutet die Mechatroniker mit dem Flooding Algorithmus.
Performance Measure	Ist dafür zuständig, die Geschwindigkeit des Algorithmus zu messen.
FloodLogger	Der Logger hält wichtige Ereignisse fest und gibt sie auf der Konsole aus, sowie, wenn es vom Benutzer festgelegt worden ist, auch eine Datei an den

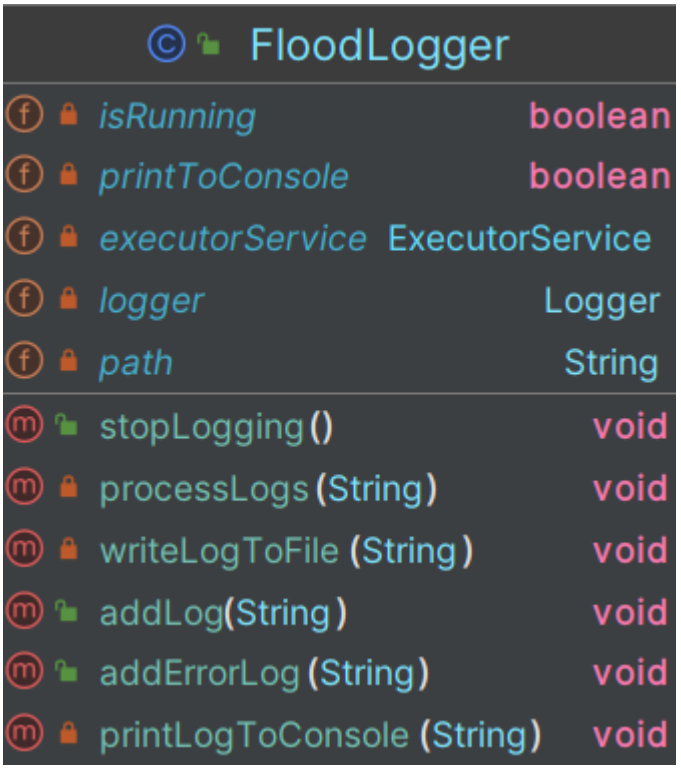
	Benutzer zurück.
ApplicationStub	Von hier aus werden die GRPC invokes gemacht, sowie ein Server erstellt welche ein Invoke an uns abfängt
Task	Eine Aufgabe, welche einfach zum Mapping benutzt wird um die Eingabe des Benutzer besser zu bearbeiten
ChefImpl	Die Implementationen der GRPC Methoden welche wir anbieten

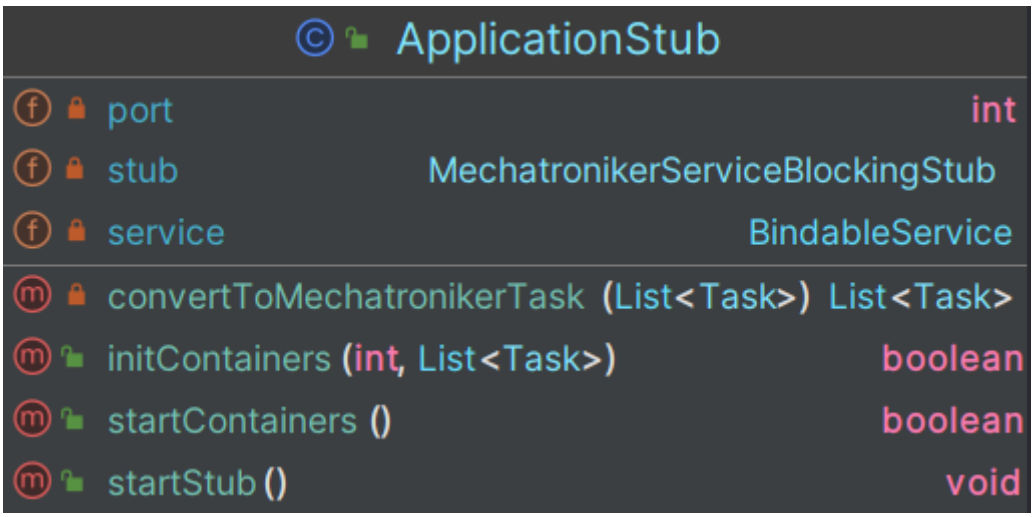
Ebene 2



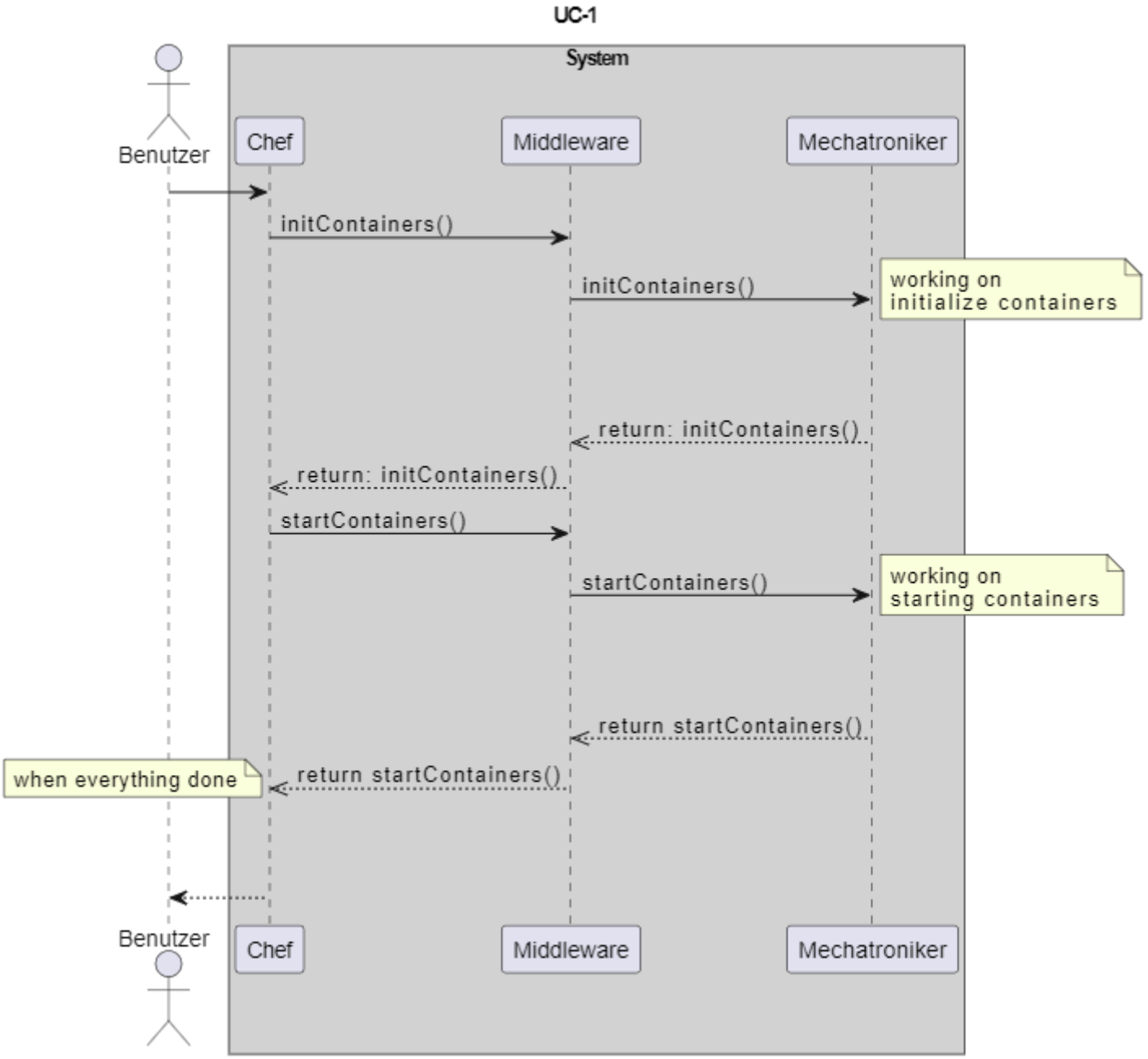



FloodMaxStart
host String
consoleLog boolean
serverPort int
ownPort int
amountOfContainers int
pathToConfig String
taskList List<Task>
checkParameters(String[]) void
mapConfig(String) void
mapTasks(String) void
main(String[]) void

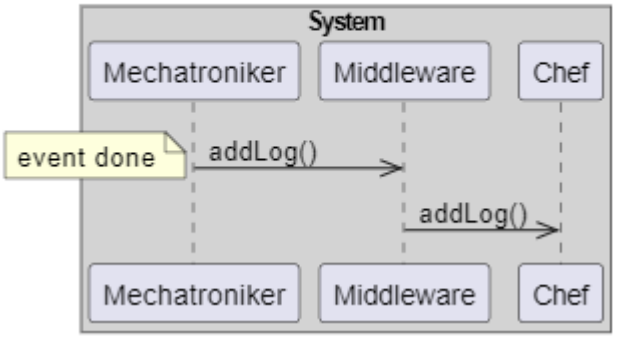

FloodLogger
isRunning boolean
printToConsole boolean
executorService ExecutorService
logger Logger
path String
stopLogging() void
processLogs(String) void
writeLogToFile(String) void
addLog(String) void
addErrorLog(String) void
printLogToConsole(String) void


ApplicationStub
port
int
stub
MechatronikerServiceBlockingStub
service
BindableService
convertToMechatronikerTask (List<Task>) List<Task>
initContainers (int, List<Task>)
boolean
startContainers ()
boolean
startStub ()
void

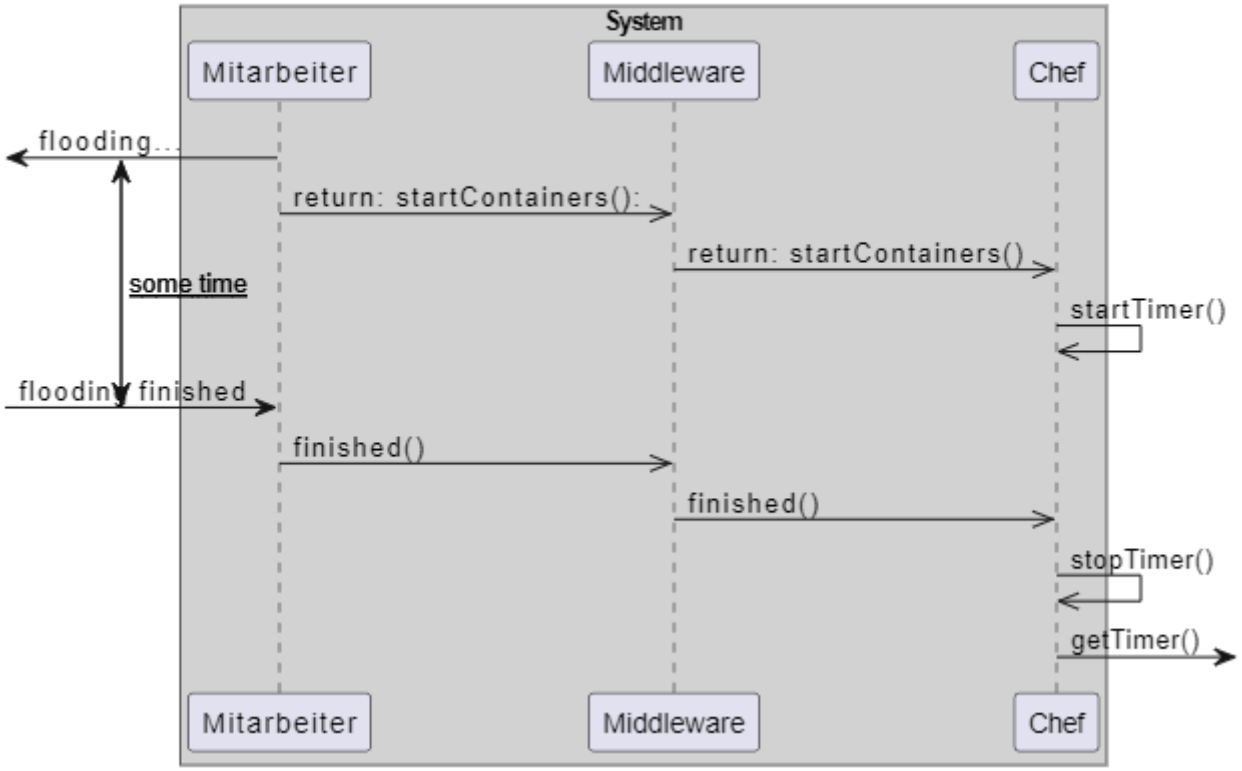
Laufzeitsicht



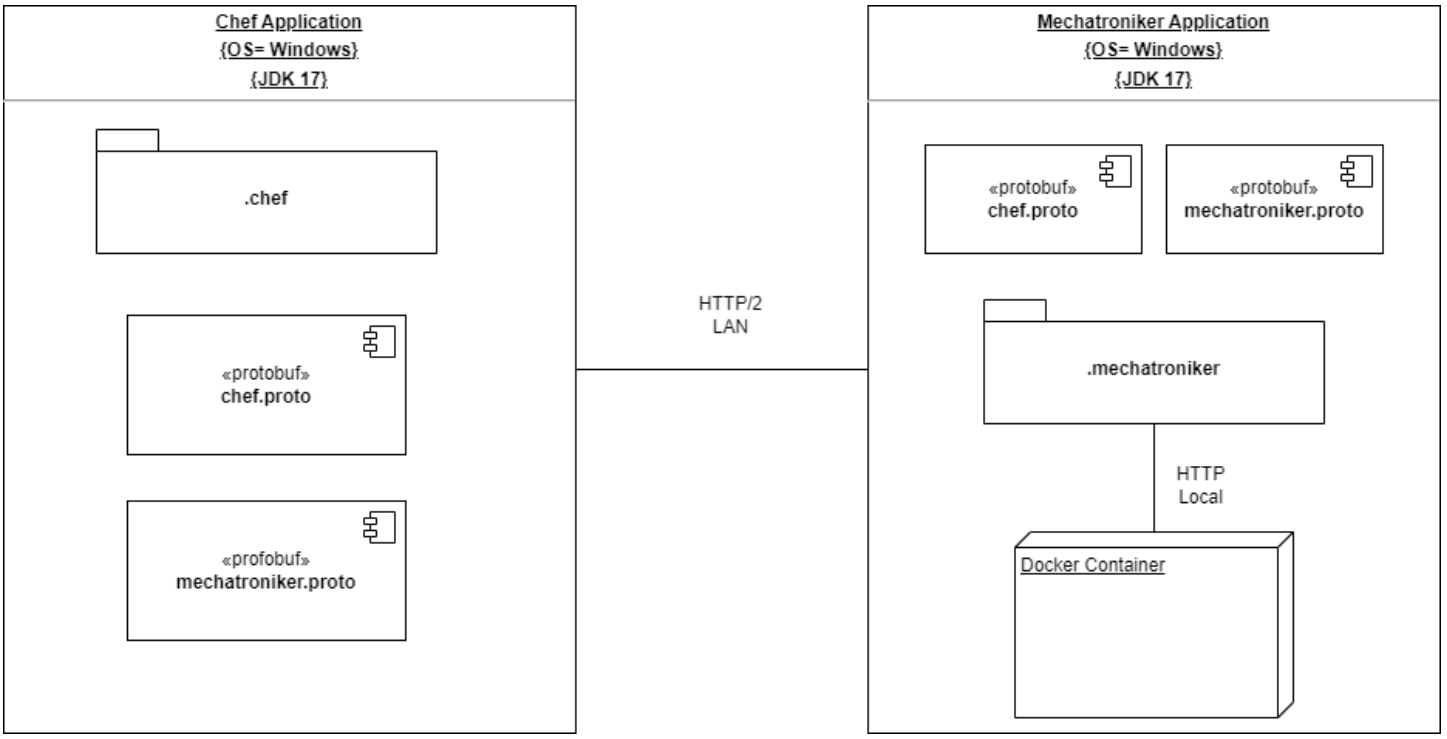
UC-3


System
Mechatroniker
Middleware
Chef
event done
addLog()
addLog()
Mechatroniker
Middleware
Chef

UC-4


System
Mitarbeiter
Middleware
Chef
flooding...
return: startContainers():
return: startContainers()
some time
startTimer()
flooding finished
finished()
finished()
stopTimer()
getTimer()
Mitarbeiter
Middleware
Chef

Verteilungssicht



Architekturentscheidungen

GRPC:

In der Mitte des Projektzeitraums haben wir uns für die Integration von gRPC als Architekturentscheidung entschieden, die sich als äußerst vorteilhaft erwiesen hat. Die Einführung von gRPC ermöglicht eine effiziente, plattformübergreifende Kommunikation zwischen verteilten Systemkomponenten und verbessert die Interoperabilität, indem sie unterschiedliche Technologien harmonisiert. Durch die Nutzung von HTTP/2 als Kommunikationsprotokoll profitieren wir von verbesserter Performance und effizienterer Ressourcennutzung. Die Verwendung von Protobuf zur Definition von Diensten und Nachrichten schafft eine klare und konsistente Schnittstelle. Darüber hinaus hat sich gRPC als robust und skalierbar erwiesen, insbesondere im Middleware-Bereich, wo automatisch generierter Code die Wartbarkeit verbessert und potenzielle Fehlerquellen minimiert. Diese Entscheidung betont nicht nur unsere Anpassungsfähigkeit an neue Technologien, sondern bietet auch eine solide Grundlage für eine effiziente und zuverlässige Kommunikation zwischen den Architekturkomponenten.

Middleware

Einführung und Ziele

Die Middleware spielt eine entscheidende Rolle bei der verteilten Bereitstellung der Anwendung über Loopback. Sie integriert RPC-Funktionalitäten, die es den verschiedenen Systemen ermöglichen, effizient miteinander zu kommunizieren und somit eine nahtlose Interaktion über das Loopback-Netzwerk zu gewährleisten.

Qualitätsziele

Qualitätsziel	**Erläuterung**
Openness	a) Systeme sollten gut definierten Schnittstellen entsprechen b) Systeme sollten leicht interoperabel sein c) Systeme sollten die Portabilität von Anwendungen unterstützen d) Die Systeme sollten leicht erweiterbar sein
Scaling	a) geographisch - Loopback oder LAN b) administrativ - kein Administrator c) Problemgröße - N- Nodes zum lösen des Problems
Distribution transparencies	a)Fehlerfälle 1)kleiner Fehler - sollte in der Visualisierung angezeigt und in die .log geschrieben werden 2)größere Fehler - führt zu einer nicht erfolgreichen Programmbeendigung

Entwurfsentscheidung

Entwurf	Erläuterung
GRPC (Remote Procedure Call)	gRPC (Remote Procedure Call) ist ein von Google

	entwickeltes Framework für die Fernaufruf Kommunikation zwischen verteilten Systemen. Es ermöglicht die nahtlose Interaktion zwischen Client- und Serveranwendungen über das Netzwerk, indem es auf dem bewährten Konzept des Remote Procedure Calls basiert. Durch die klare Trennung von Schnittstellenbeschreibungen und Implementierungsdetails fördert gRPC eine lose Kopplung zwischen den Komponenten. Dies erleichtert die Entwicklung, Wartung und Skalierung von verteilten Systemen, indem es effiziente und plattformübergreifende Kommunikation ermöglicht.
zweistufen Architektur	Jedes Modul kümmert sich um eine spezifische Aufgabe oder Funktion der Anwendung. Durch die klare Modularität wird der Code wartbarer, erweiterbar und leichter zu verstehen.

Kontextabgrenzung

Fachlicher Kontext

Nachbarsystem /Benutzer	**Beschreibung**	**Input**	**Output**
Chef	Die Benutzerschnittstelle und es ist zuständig für das Logging sowie die Zeitmessung des Algorithmus.	HTTP/2 Request	HTTP/2 Response
Mechatroniker	Das Mitarbeiter-System ist für die Orchestrierung der Container zuständig, welche den FloodMax Algorithmus anwenden, um auf einen Konsens zu kommen.	HTTP/2 Request	HTTP/2 Response

Lösungsstrategie

Verweis auf JavaDoc im Gitlab.