DSE22Gruppe10Projektplan

Diego Krupitza, Jan Müller, Kian Pouresmaeil

5. April 2022

Inhaltsverzeichnis

1 Projektzielplan

Ziel dieses Projektes ist die Entwicklung eines Softwaresystems, welches eine Kommunikation zwischen autonomen Fahrzeugen und Verkehrsinfrastruktur simuliert. Dabei soll eine intelligente Verkehrsflusssteuerung auf Basis eines Algorithmus umgesetzt werden. Die Umsetzung soll auf Basis eines verteilten Systems erfolgen, wobei einzelne Komponenten voneinander entkoppelt implementiert werden sollen. Dafür sollen Komponenten mithilfe von REST und einer „MoM“ kommunizieren.

Folgende Projektziele werden realisiert:

- Es ist möglich eine Verkehrssituation mit bis zu vier Fahrzeugen und vier Ampeln zu simulieren.
- Einzelne Komponenten sind voneinander entkoppelt und sind daher horizontal skalierbar.
- Eine Web-Applikation erlaubt Benutzerinnen und Benutzern das Bedienen des Systems über einen Browser.
- Benutzerinnen und Benutzern ist es möglich die Anzahl an Fahrzeugen und Ampeln einer Simulation sowie deren Startzustände zu konfigurieren.
- Für die Geschwindigkeitsberechnung eines Fahrzeuges berücksichtigt der Algorithmus nur Ampeln, in deren Scanreichweite sich das betroffene Fahrzeug befindet.
- Der Algorithmus arbeitet mit realistischen Geschwindigkeiten in dem Bereich von 0 bis 130 km/h. Ein Stillstand (0 km/h) erfolgt nur, wenn ein Fahrzeug vor einer roten Ampel halten muss.
- Die Simulation ist so vereinfacht, dass nur eine Koordinate relevant ist.

Folgende Projektziele werden nicht realisiert:

- Der implementierte Algorithmus entspricht einer optimalen Lösung in Bezug auf Zeit- und Speicherkomplexität.
- Komponenten verwenden Autorisierung und Authentifizierung um Kommunikation abzusichern.
- Der Algorithmus findet die global optimale Lösung, indem auch Ampeln außerhalb der Scanreichweite berücksichtigt werden.

- Das System simuliert realistische Beschleunigungs- und Bremsvorgänge.
- Die Web-Applikation ist für Mobilgeräte optimiert.
- Das System verfügt über Ausfallsicherheit.
- Das Service Mesh wird visualisiert.
- Eine gelbe Ampelphase wird bei der Simulation berücksichtigt.

2 Projektorganigramm

2.1 Rollen- und Aufgabenverteilung

Unabhängig von den zugeteilten Rollen arbeiten alle Teammitglieder an allen Aspekten der Applikation.

- Diego Krupitza ist *Projektleiter* und *Softwareentwickler*.
- Kian Pouresmaeil ist *Softwarearchitekt* und *-entwickler*.
- Jan Müller ist *Frontend-Architekt* und *Softwareentwickler*.

2.2 Verantwortlichkeiten

Alle Teammitglieder sind im Projekt gleichgestellt. Die Rollenverteilung gilt als Grundlage für die Verteilung der Verantwortlichkeiten.

Beispielsweise ist der Frontend-Architekt Jan Müller verantwortlich für die vereinbarte *Best-Practice-Implementierung* des Frontends. Unabhängig von den Rollen ist jedes Teammitglied für die Implementierung aller Schichten verantwortlich. Für die interne Koordination und Teamsteuerung ist Diego Krupitza verantwortlich. Die Hauptaufgabe darin besteht, den Entwicklungsverlauf zu überwachen und die Arbeit der Teammitglieder zu koordinieren.

Aufgrund der flachen Struktur sind alle Teammitglieder gleichgestellt und es gibt keine Projektinterne Hierarchie.

3 Work Breakdown Structure (WBS)

Die folgende Aufzählung enthält alle Arbeitspakete. Die Zahl hinter dem Titel eines Arbeitspaketes entspricht einer Aufwandsschätzung in Story Points (SP).

1. Aufsetzen eines Netflix Eureka Servers (1 SP)
   In diesem Arbeitspaket wird eine Netflix Eureka Server Instanz aufgesetzt, damit diese in späterer Folge für Service Registrierung sowie Discovery verwendet werden kann.

2. Aufsetzen eines RabbitMQ Servers (1 SP)
   In diesem Arbeitspaket wird mithilfe von Docker Compose eine RabbitMQ Instanz aufgesetzt, welche als „MoM“ dient.

3. Aufsetzen einer MongoDB Instanz (1 SP)
   In diesem Arbeitspaket wird eine MongoDB Instanz für die Entwicklungsumgebung aufgesetzt.

4. Konfigurieren eines Spring Cloud Gateways (2 SP)
   In diesem Arbeitspaket wird ein Gateway implementiert, welcher Anfragen der Web-Applikation an die zugehörigen Dienste weiterleitet. Beim Starten des Dienstes soll eine Registrierung beim Eureka Server erfolgen.

5. EntityService - Aufsetzen (2 SP)
   In diesem Arbeitspaket wird die Grundstruktur des EntityServices auf Basis von TypeScript implementiert damit in späterer Folge darauf aufgebaut werden kann. Das EntityService soll sich nach dem Starten beim Eureka Server registrieren. Es soll ein Dockerfile zum Bauen eines Docker Images erstellt werden.

6. EntityService - Registrierung von Fahrzeugen (2 SP)
   In diesem Arbeitspaket soll das Empfangen und Persistieren von relevanten Fahrzeuginformationen auf Basis der „MoM“ Queue „car“ implementiert werden.

7. EntityService - Registrierung von Ampeln (2 SP)
   In diesem Arbeitspaket soll das Empfangen und Persistieren von relevanten Ampelinformationen auf Basis der „MoM“ Queue „traffic-light“ implementiert werden.

8. EntityService - Implementieren eines Endpunktes zum Lesen von Datensätzen (2 SP)
   In diesem Arbeitspaket sollen REST-Endpunkte implementiert werden, welche das Lesen von Ampel- und Fahrzeuginformationen ermöglicht. Es soll insbesondere möglich sein Daten einzelner Entitäten über deren IDs zu lesen.

9. EntityService- Implementieren eines Endpunktes zum Durchführen von geospatial Queries zu Ampelpositionen (2 SP)
   In diesem Arbeitspaket sollen REST-Endpunkte implementiert werden, welche die Ergebnisse von geospatial Queries zu den Standorten von Ampeln liefern.

10. EntityService - Zurücksetzen (1 SP)
    In diesem Arbeitspaket wird ein REST-Endpunkt implementiert, welcher es ermöglicht den gesamten Zustand des Dienstes zurückzusetzen.

11. TrackingService - Aufsetzen (2 SP)
    In diesem Arbeitspaket wird die Grundstruktur des TrackingService auf Basis von TypeScript implementiert damit in späterer Folge darauf aufgebaut werden kann. Der TrackingService soll sich nach dem Starten beim Eureka Server registrieren. Es soll ein Dockerfile zum Bauen eines Docker Images erstellt werden.

12. TrackingService - Anbindung an „MoM“ für Ampeln (2 SP)
    In diesem Arbeitspaket wird eine Queue „traffic-light-state“ von RabbitMQ gelesen und bei eintreffenden Nachrichten die Datenbank aktualisiert. Ampeln senden neben ihrem Status zusätzlich die verbleibende Zeit bis zur nächsten Statusänderung.

13. TrackingService - Anbindung an „MoM“ für Fahrzeuge (2 SP)
    In diesem Arbeitspaket wird eine Queue „car-state“ von RabbitMQ gelesen und bei eintreffenden Nachrichten die Datenbank aktualisiert. Ein Fahrzeug sendet Standort, Fahrtrichtung, sowie seine aktuelle Geschwindigkeit.

14. TrackingService - Lesen von Daten (2 SP)
    In diesem Arbeitspaket werden REST-Endpunkte implementiert, welcher das Lesen aller Fahrzeugpositionen sowie Ampelzustände ermöglicht. Insbesondere soll es möglich sein alle Zustände sowie den aktuellsten Zustand einer Entität über deren ID zu lesen.

15. TrackingService - Zurücksetzen (1 SP)
    In diesem Arbeitspaket wird ein REST-Endpunkt implementiert, welcher es ermöglicht den gesamten Zustand des Dienstes zurückzusetzen. Dafür sollen alle persistierten Daten gelöscht werden.

16. SimulatorService (Endpoint) - Aufsetzen (3 SP)
    In diesem Arbeitspaket wird der SimulatorService aufgesetzt. Bei Star-

tup soll sich der Dienst sich bei Eureka registrieren. Neben einem Worker, welcher für die Verwaltung der Simulationsentitäten zuständig ist, soll es einen REST-Endpunkte und „MoM“ Anbindungen zur externen Kommunikation geben.

17. SimulatorService (Endpoint) - Simulation von Ampeln (2 SP)
    In diesem Arbeitspaket wird die Simulation von Ampeln implementiert. Ampeln wechseln periodisch zwischen den Zuständen „rot“ und „grün“ und senden bei jedem Wechsel ihren aktuellen Zustand an die „MoM“ Queue „traffic-light-state“.

18. SimulatorService (Endpoint) - Simulation von Fahrzeugen (3 SP)
    In diesem Arbeitspaket wird die Simulation von Fahrzeugen implementiert. Fahrzeuge fahren mit einer gegebenen Geschwindigkeit und melden periodisch ihren aktuellen Zustand an die „MoM“ Queue „car-state“. Zudem wird die Queue „car-speed“ auf eingehenden Nachrichten beobachtet und bei Bedarf die Geschwindigkeit von Fahrzeugen angepasst.

19. SimulatorService (Endpoint) - Szenario erstellen (2 SP)
    In diesem Arbeitspaket wird ein REST-Endpunkt zum Erstellen und Starten von Szenarios implementiert. Dieser soll alle relevanten Informationen empfangen und darauf aufbauend Simulationen für Fahrzeuge und Ampeln starten.

20. SimulatorService (Endpoint) - Zurücksetzen (2 SP)
    In diesem Arbeitspaket wird ein REST-Endpunkt implementiert, welcher beim Aufruf die aktuelle Simulation terminiert.

21. SimulatorService (Endpoint) - Flux Kompensator (2 SP)
    In diesem Arbeitspaket wird ein Zeitraffer für Simulationen implementiert. Der Faktor einer Zeitraffung soll beim Erstellen eines Szenarios anpassbar sein.

22. FlowControlService - Aufsetzen (2 SP)
    In diesem Arbeitspaket wird die Grundstruktur des Spring Boot FlowControlService implementiert damit in späterer Folge darauf aufgebaut werden kann. Der FlowControlService soll sich nach dem Starten beim Eureka Server registrieren. Es soll ein Dockerfile zum Bauen eines Docker Images erstellt werden.

23. FlowControlService - Algorithmus (5 SP)
    In diesem Arbeitspaket soll der Algorithmus zur Optimierung des Verkehrsflusses implementiert werden. Dazu soll der FlowControlService bei der „MoM“ auf Nachrichten einer Queue „car-state“ achten, welche Standortaktualisierungen von Fahrzeugen melden. Bei Eintreffen solcher Nachrichten soll die optimale Geschwindigkeiten betroffener Fahrzeuge ermittelt werden, sodass diese noch bei „grün“ die Ampel passieren können beziehungsweise, falls möglich, vor der Ampel nicht zum Stillstand kommen müssen. Die so ermittelten Geschwindigkeiten sollen über die „MoM“ Queue „car-speed“ an die betroffenen Fahrzeuge gesendet werden. Es muss eine zulässige Höchstgeschwindigkeit implementiert werden.

24. Cockpit - Aufsetzen (1 SP)
    In diesem Arbeitspaket wird die grundlegende Einrichtung der Web-Applikation durchgeführt. Es soll ein Dockerfile zum Bauen eines Docker Images erstellt werden.

25. Cockpit - Simulationen starten (1 SP)
    In diesem Arbeitspaket wird ein Formular zum Starten von Simulationen mit konfigurierbaren Parametern implementiert.

26. Cockpit - Simulationen visualisieren (3 SP)
    In diesem Arbeitspaket wird eine Visualisierung von laufenden Simulationen in der Web-Applikation implementiert. Es sollen die Positionen von Fahrzeugen und Ampeln in einer schematischen Karte angezeigt werden

27. Cockpit - Informationen von Entitäten anzeigen (2 SP)
    In diesem Arbeitspaket wird eine Anzeige von Informationen über Fahrzeuge und Ampeln implementiert.

28. Swagger Dokumentation (3 SP)
    In diesem Arbeitspaket wird der API Gateway mittels Swagger dokumentiert.

29. Deployment auf der GCP (5 SP)
    In diesem Arbeitspaket soll das Deployment, welches während der Entwicklung auf Basis von Docker Compose und Minikube stattfindet, auf die Google Cloud Platform migriert werden.

4 Aufwandsschätzung

Auf Basis der WBS mit einer SP Summe von 70 lässt sich ein Zeitaufwand von ungefähr 33 Stunden pro Person schätzen. Der Gesamtaufwand für die Implementierung beläuft sich somit auf 99 Stunden.

5 Ressourcenplanung und Meilensteinplanung

5.1 Verteilung der Arbeitspakete

Ziel ist es bis zum 31.5.2022 das Projekte mit Ausnahme der Dokumentation abzuschließen. Daher sollen bis zum 1.5.2022 die ersten 16 Arbeitspakete und bis zum die restlichen Arbeitspakete 31.5 abgeschlossen werden. Die Arbeitspakete werden nicht im Voraus aufgeteilt, sondern in einem Backlog hinterlegt. Teammitglieder nehmen sich dann einzelne Pakete aus diesem Backlog heraus um sie gemäß dem Kanban Prinzip abzuarbeiten.

5.2 Fertigstellungstermine

Nach der Fertigstellung der Implementierung am 31.5.2022 werden bis zum 17.6.2022 die Dokumentation sowie die restlichen Deliverables fertiggestellt. Dabei ist der 16.6.2022 als interne Deadline festgelegt.

6 Technische Planung

Ein Eureka stellt neben einer Registrierung für einzelne Dienste sowie darauf aufbauender Service Discovery auch Statusüberwachung bereit.

Als „MoM“ für eine asynchrone Kommunikation wird RabbitMQ verwendet.

Ein Spring Cloud Gateway wird als Einstiegspunkt für externe Kommunikation dienen. Dafür werden Anfragen an zuständige Komponenten weitergeleitet und Load-Balancing durchgeführt.

Zum Persistieren von Daten wird MongoDB verwendet, da die Datenbestände keine Relationen zwischen Entitäten aufweisen und sie sich somit für dokumentbasierte Datenbanken eignen.

EntityService und TrackingService werden mit Node.js, TypeScript sowie Fastify implementiert.

FlowControlService und SimulatorService werden hingegen mit Spring Boot entwickelt. Dabei steht insbesondere die simple Konfiguration im Vordergrund.

Die Web-Applikation wird mit Vue 3 und Vite entwickelt, um den Anteil an Boilerplate zu minimieren und schnelles Iterieren während der Entwicklung zu ermöglichen.

Während der Entwicklung werden Docker Compose und minikube verwendet um das System lokal in Betrieb zu nehmen.

7 GCP Budget Schätzung

Für die GCP Budget Schätzung wurde der Google Cloud Pricing Calculator verwendet[1]. Dabei wurden neun Standard-Instanzen am Standort Frankfurt für 360 Stunden pro Monat angegeben. Die errechneten Kosten belaufen sich auf 22,03 USD[2].

[1] Siehe `https://cloud.google.com/products/calculator/#id=e4dfe221-3d62-4d60-98ca-cc0d06625a63`

[2] Stand 4. April 2022