Webanwendung zur Visualisierung von rettungsrelevanten Daten zur Einsatzunterstützung

Software-Entwicklungspraktikum (SEP)
Sommersemester 2023

Technischer Entwurf

Auftraggeber
Technische Universität Braunschweig
Peter L. Reichertz Institut für Medizinische Informatik
Prof. Dr. Thomas Deserno
Mühlenpfordtstraße 23
38106 Braunschweig

Betreuer: Viktor Sobotta

Auftragnehmer:

Name	E-Mail-Adresse
Mohamed Wassim Chebili	m.chebili@tu-braunschweig.de
Omar Farouk Khayat	o.khayat@tu-braunschweig.de
Jonas Stepanik	j.stepanik@tu-braunschweig.de
Azhar Rahadian	a.rahadian@tu-braunschweig.de
Kacem Abdennabih	k.abdennabih@tu-braunschweig.de
Torben Oelerking	t.oelerking@tu-braunschweig.de
Qiyue Zhang	qiyue.zhang@tu-braunschweig.de

Braunschweig, 28. Juni 2023

Bearbeiterübersicht

Kapitel	Autoren	Kommentare
1	Omar Farouk Khayat,Mohammed Wassim Chebili	keine
1.1	Omar Farouk Khayat,Mohammed Wassim Chebili	keine
2	Jonas Stepanik, Torben Oelerking	keine
2.1	Jonas Stepanik, Torben Oelerking	keine
2.2	Jonas Stepanik, Torben Oelerking	keine
3	Omar Farouk Khayat,Mohammed Wassim Chebili	keine
3.1	Omar Farouk Khayat,Mohammed Wassim Chebili	keine
3.2	Omar Farouk Khayat,Mohammed Wassim Chebili	keine
3.3	Azhar Rahadian	keine
4	Omar Farouk Khayat,Mohammed Wassim Chebili	keine
5	Jonas Stepanik, Torben Oelerking, Kacem Abdennabih	keine
5.1	Torben Oelerking	keine
5.2	Kacem Abdennabih	keine
5.3	Kacem Abdennabih	keine
5.4	Jonas Stepanik	keine
6	Kacem Abdennabih	keine
6.1	Kacem Abdennabih	keine

6.2	Kacem Abdennabih	keine
7	Qiyue Zhang	keine
8	Zusammen	keine
9	Qiyue Zhang	keine
9.1	Qiyue Zhang	keine
9.2	Qiyue Zhang	keine
9.3	Qiyue Zhang	keine
10	Zusammen	keine

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

ieses Dokument enthält eine umfassende und detaillierte Beschreibung der verschiedenen Funktionen und ihrer Implementierung. Es dient als Erweiterung des bereits vorhandenen Fachentwurfs und liefert zusätzliche Informationen, um ein tieferes Verständnis des Systems zu ermöglichen. Ein zentraler Aspekt ist die Vorstellung der Architektur unseres Systems. Dabei wird das gesamte System in verschiedene Komponenten aufgeteilt, die jeweils spezifische Aufgaben und Verantwortlichkeiten haben. Des Weiteren wird die Verteilung unseres Projekts betrachtet. Dies beinhaltet die Aufteilung und Zuweisung der Komponenten auf verschiedene Hardware- oder Netzwerkumgebungen. Ein wichtiger Teil dieses Dokuments ist die ausführliche Beschreibung der Implementierung jeder einzelnen Komponente. Hier werden die zugrunde liegenden Technologien, Frameworks oder Bibliotheken erläutert, die für die Umsetzung verwendet wurden, und es werden Paket-/Klassendiagramme vorgestellt. Darüber hinaus enthält das Dokument ein Kapitel, das die Änderungen und Erweiterungen gegenüber dem ursprünglichen Fachentwurf auflistet. Abschließend werden die Komponenten, welche spezifische Kriterien erfüllt haben, präsentiert. Durch die Identifizierung und Erläuterung dieser Kriterien wird deutlich, wie die einzelnen Komponenten zur Gesamtfunktionalität des Systems beitragen.

1.1 Projektübersicht

Unser Projekt hat den Zweck, relevante Daten eines Unfalls darzustellen und den Zustand der Fahrzeuge nach dem Unfall auf einer Karte zu visualisieren. Aus Sicht des Nutzers erfolgt zunächst die Eingabe einer Unfall-ID. Diese ID wird an den Webserver übermittelt. Wenn ein Unfall mit dieser ID existiert, werden die Daten und das 3D-Mesh an die Webanwendung übertragen. Andernfalls wird der Nutzer aufgefordert, eine andere ID einzugeben. Wenn die eingegebene ID korrekt ist, wird eine Karte angezeigt, auf der die Fahrzeuge des Unfalls dargestellt werden. Wenn der Nutzer sein Maus auf ein Fahrzeug bewegt, werden die Daten des Fahrzeugs in einem Overlay angezeigt. Durch Klicken auf das Fahrzeug werden die auf die Insassen einwirkenden Kräfte angezeigt.
Dem Nutzer wird zusätzlich ein Link zum Anzeigen der Rettungskarte eines Fahrzeuges bereitgestellt.
Der Nutzer hat die Möglichkeit, durch Drücken der Home-Taste zur Startseite zurückzukehren und eine weitere ID einzugeben. Dieses Verhalten wird in dieser Abbildung veranschaulicht.

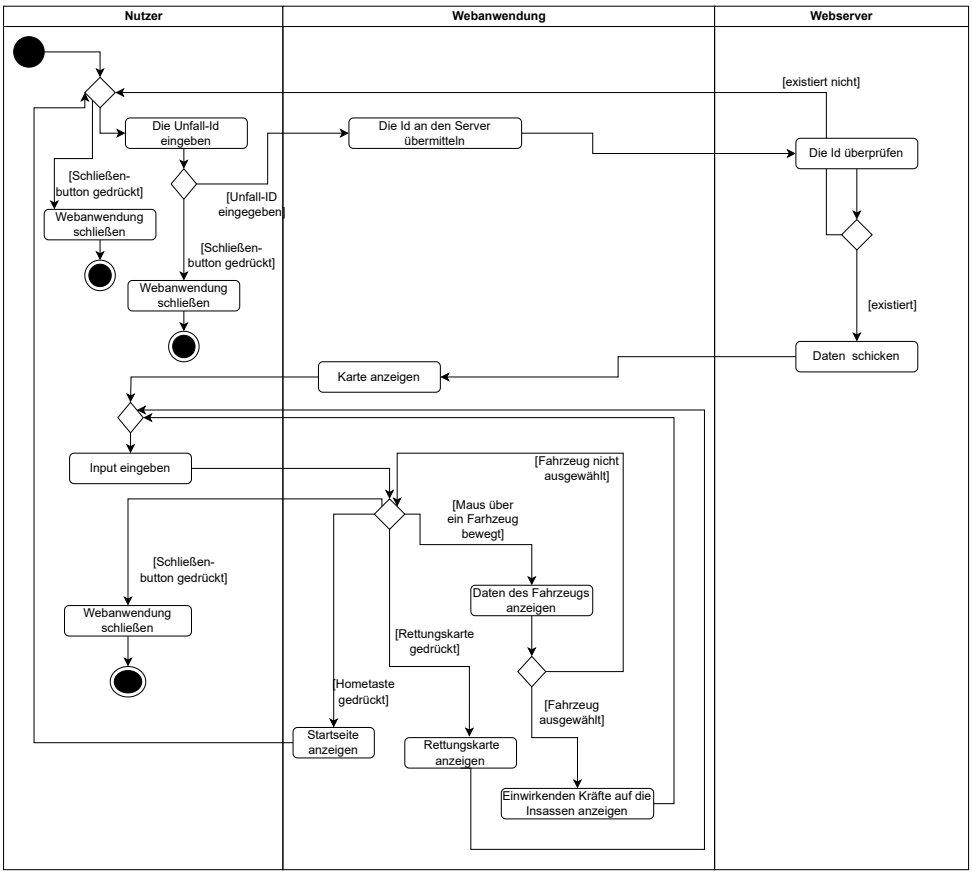

Abbildung 1.1: Aktivitätsdiagramm aus der Nutzersicht

Der Webserver empfängt eine ISAN und überprüft das Format der ISAN. Falls das Format nicht korrekt ist, sendet der Webserver eine Fehlermeldung als Antwort zurück. Andernfalls werden die Daten aus der ISAN extrahiert. Wenn dieser Unfall geografisch und zeitlich nahe an einem gespeicherten Unfall liegt, werden die Daten mit diesem Unfall gruppiert. Andernfalls werden sie eigenständig behandelt.
Sobald die Daten extrahiert und vorbereitet sind, fordert der Webserver eine Simulation beim Simulations-Server an. Der Simulations-Server führt die Simulation durch und sendet dem Webserver die Simulationsergebnisse als Rückmeldung. Die erhaltenen Daten und die Simulationsergebnisse werden anschließend gespeichert und stehen der Webanwendung zur Verfügung. Dieses Verhalten wird in dieser Abbildung veranschaulicht.

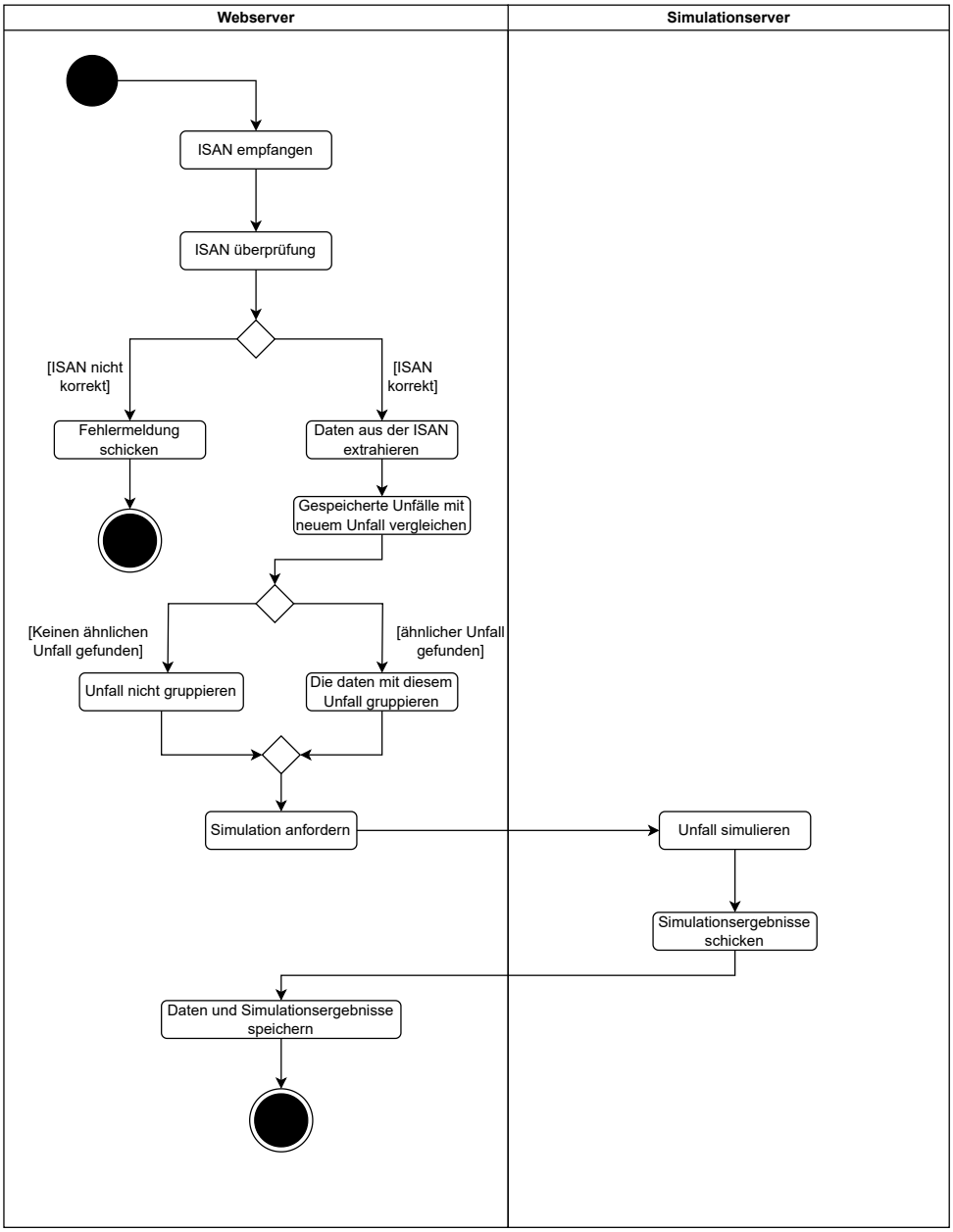

Abbildung 1.2: Aktivitätsdiagramm beim hinzufügen einen neuen Unfall

1.2 Projektdetails

In diesem Abschnitt werden spezifische Aspekte des Projekts detaillierter erläutert und mit Hilfe von Grafiken und Beispielen veranschaulicht. Durch diese zusätzliche Darstellung wird ein tieferes Verständnis der jeweiligen Themenbereiche ermöglicht.

1.2.1 ISAN Format

Die Verwendung des ISAN-Formats ermöglicht eine standardisierte und einfache Übermittlung von Unfalldaten. Der ISAN besteht aus einer Zeichenkette, die weiterverarbeitet wird. Im Folgenden wird das allgemeine Format des ISAN dargestellt:

<Längengrad>;<Breitengrad>;<Uhrzeit>;<Datum>;<Kraft>;<Geschwindigkeit>;<Hsn>;<Tsn>;<Kfz-Kennzeichen>;<Anzahl der Insassen>

- **Längengrad und Breitengrad** bestimmen Die Position des Autos. Diese werden ohne Komma dargestellt. Zuerst wird das Vorzeichen (+/-) angegeben, gefolgt von zwei vor dem Komma für den Breitengrad und drei Zahlen für den Längengrad. Anschließend folgen fünf Dezimalstellen.
- **Uhrzeit** wird im Format HHMMSS dargestellt.
- **Datum** wird im Format YYYYMMDD dargestellt.
- **Kraft**, die im Moment des Unfalls ausgeübt wird und in kN ohne Nachkommastellen angegeben ist.
- **Geschwindigkeit**, die das Fahrzeug zum Zeitpunkt des Unfalls hatte in km/h. Die Geschwindigkeit kann maximal 3 Ziffern haben.
- **Hsn und Tsn** ergeben zusammen das Modell des Autos. Die HSN wird in 4 Zeichen dargestellt, während die TSN in 3 Zeichen dargestellt wird.
- **Das KFZ-Kennzeichen** darf höchstens 8 Zeichen umfassen.
- **Anzahl der Insassen** umfasst eine Ziffer.

Hier ist ein Beispiel für eine ISAN für einen Unfall mit den folgenden Daten:
Am 01.06.2023 um 11:30:26 Uhr ereignete sich ein Unfall in der Hambugerstraße in Braunschweig. Das betroffene Fahrzeug war ein Golf 6 mit dem KFZ-Kennzeichen "BSAO123". Während des Unfalls wurde das Fahrzeug einer Kraft von 30 kN ausgesetzt, und seine Geschwindigkeit betrug zum Zeitpunkt des Unfalls 60 km/h. Zu diesem Zeitpunkt befanden sich insgesamt 4 Personen

im Fahrzeug.

ISAN:"+01052038;+5227749;113026;20230601;30;60;0603;ANY;BSAO123;4"

1.2.2 BeamNGpy Simulation

Um mit BeamNGpy einen Unfall zu simulieren, wird die Kraft, die Geschwindigkeit, Hsn unf Tsn an den Simulations-Server gesendet. Der Simulations-Server führt die Simulation durch und sendet das 3D-Mesh zurück. Das 3D-Mesh wird als GLTF-Datei dargestellt und repräsentiert das simulierte Auto.

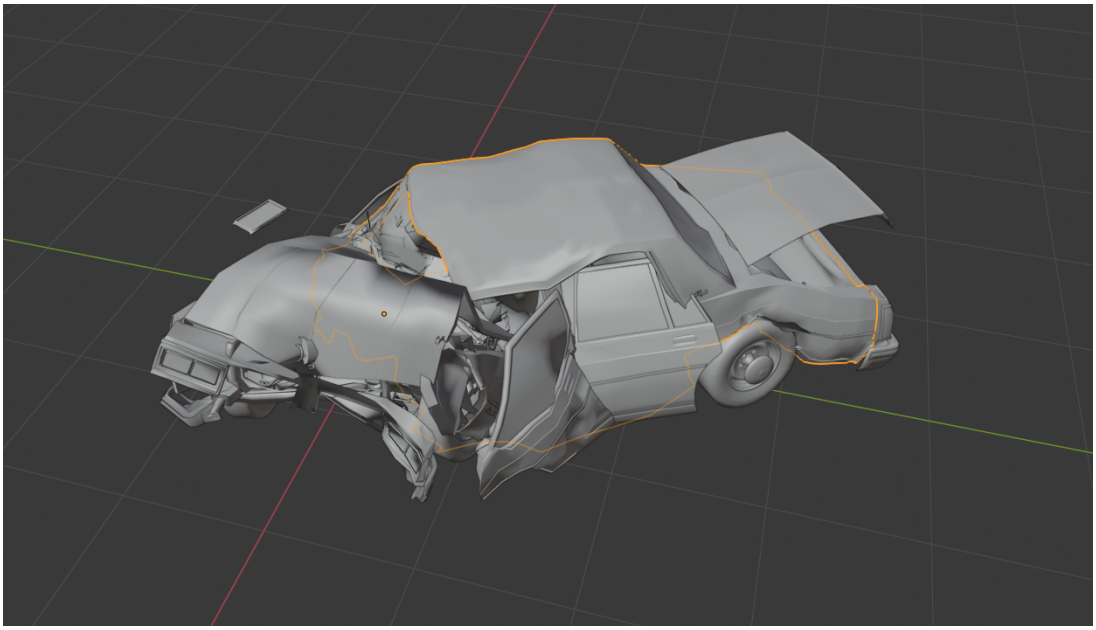

Abbildung 1.3: Ein simulierter Unfall

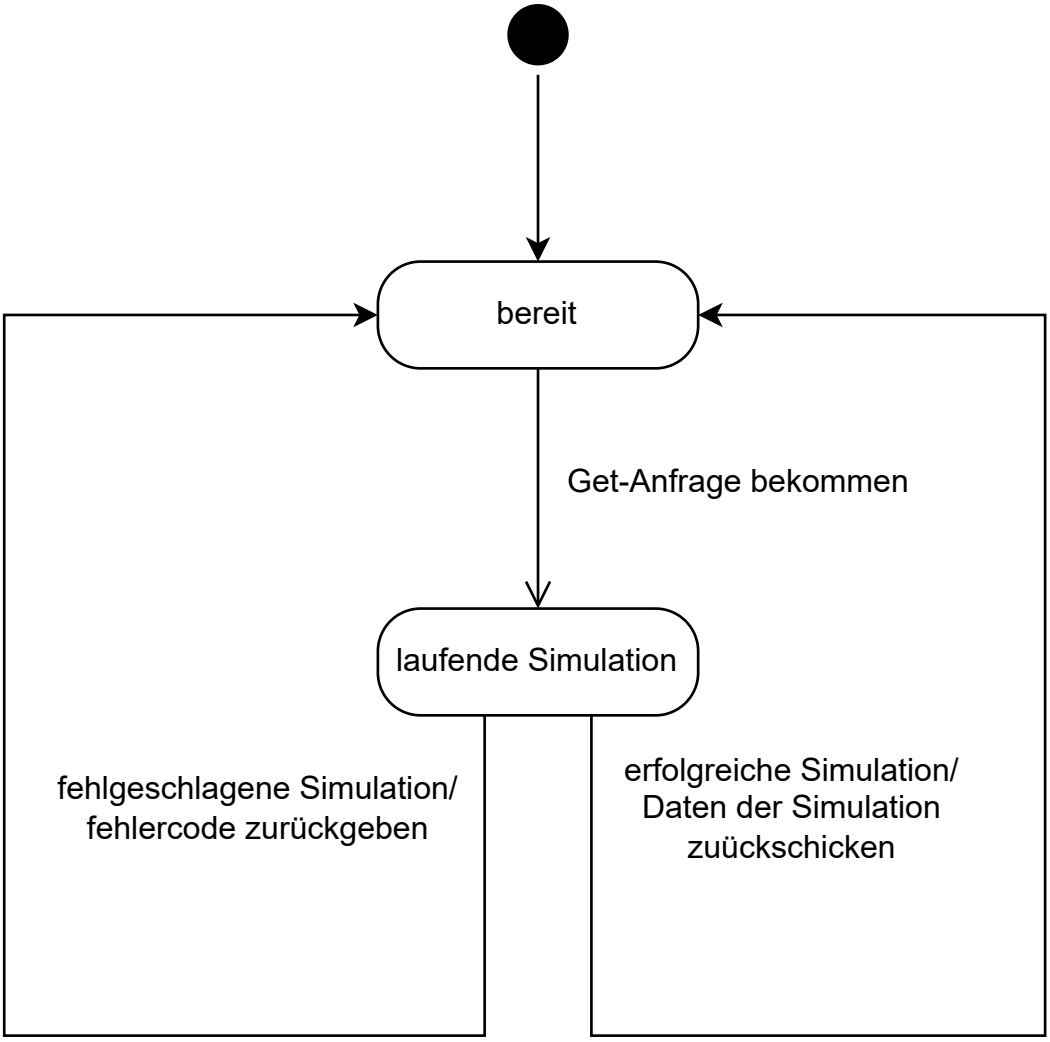

Abbildung 1.4: Zustanddiagramm der Simulation

In diesem Zustandsdiagramm kann der Verlauf unseres Simulationsservers veranschaulicht werden. Zu Beginn befindet sich unser Server im Zustand "Bereit". Wenn eine GET-Anfrage eingeht, die eine Simulation anfordert, wird die Simulation in BeamNGpy ausgeführt und der Server befindet sich im Zustand "Laufende Simulation". In diesem Zustand wartet er auf BeamNGpy. Falls die Simulation fehlschlägt, sendet der Server einen Fehlercode zurück. Andernfalls sendet er die angeforderten Daten. In beiden Fällen kehrt er zum Zustand "Bereitßurück.

1.2.3 REST APIs

Die Kommunikation zwischen unseren Komponenten erfolgt über HTTP Requests. Hier wird das Format und die Methoden dieser Anfragen weiter erläutert.

HTTP Request zum Hinzufügen eines neuen Unfalls: Um einen Unfall zu melden, wird eine POST-Request an die URL "<base-web-server-url>/accidents" geschickt. So sieht diese POST-Request aus:

```
{
    "ISAN": "+01052038;+5227749;113026;20230601;30;60;0603;ANY;BSAO123;4"
}
```

HTTP Request zum anfordern einer Simulation: Um einen Unfall zu simulieren, wird ein GET-Request an die URL "<base-simulation-server-url>/simulateforce=<force in kN> &speed=<speed in km/h>&hsn=<hsn>&tsn=<tsn>" geschickt. Es kommt dann eine Response mit dem simulierten Model.
So sieht diese Response aus:

```
{
    "mesh": <GLTF model>
}
```

HTTP Request zum Anfordern eines Unfalls: Um die Daten eines Unfalls zu bekommen, wird ein GET-Request an die URL "<base-web-server-url>/accidents?id=<id>" geschickt. Es kommt dann eine Response zurück, welche alle Informationen zu den einzelnen Autos des Unfalls zurückgibt. So sieht diese Response aus:

```
{
  "time": "20:18:28",
  "date": "07.12.2022",
  "id": 22,
  "cars": [
    {
      "latitude": 52.26666,
      "longtitude": 10.51666,
      "mesh": <GLTF Datei>,
      "registrationNumber": "BSIO123",
      "hsnTsn": "0005/173",
      "numberOfPassengers": 3,
      "force": 20,
      "speed": 40
    },
    {
      "latitude": 52.266623,
      "longtitude": 10.516933,
      "mesh": <GLTF Datei>,
      "registrationNumber": "BSAU1988",
      "hsnTsn": "0005/625",
      "numberOfPassengers": 2,
      "force": 40,
      "speed": 70
    }
  ]
}
```

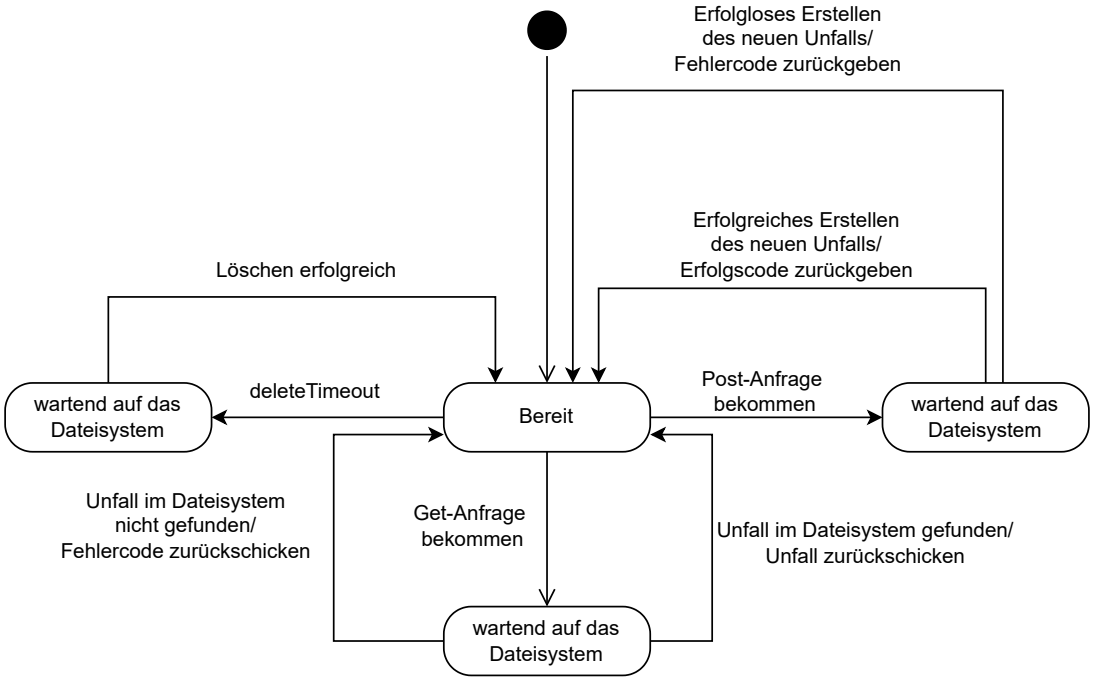

Abbildung 1.5: Zustanddiagramm des Webservers

In diesem Zustandsdiagramm kann der Verlauf des Webservers veranschaulicht werden. Zu Beginn befindet er sich im Zustand "Bereit". Es können drei Ereignisse auftreten.

Wenn eine GET-Anfrage zur Anforderung eines Unfalls eingeht, geht der Webserver in den Zustand "Wartend auf das Dateisystem". Wenn die Datei gefunden wird, wird der Unfall zurückgeschickt. Andernfalls wird ein Fehlercode zurückgegeben. In beiden Fällen kehrt der Server zum Ausgangszustand zurück.

Wenn der Server im Zustand "Bereitïst und das Ereignis "Delete Timeoutëintritt (dieses Ereignis wird in regelmäßigen Zeitintervallen aufgerufen), muss er auf das Löschen der alten Unfälle warten und kehrt dann zum Zustand "Bereitßurück.

Das dritte Ereignis ist eine POST-Anfrage, die eine ISAN (International Standard Accident Number) liefert. Hier wartet der Server darauf, dass eine Datei erzeugt wird. Abhängig vom Ergebnis der Erzeugung einer Datei wird ein entsprechender Code zurückgegeben. Beispielsweise könnte der Server den Code Ërfolgreich erzeugtöder "Fehler bei der Erzeugungßurückgeben, um den Zustand des erzeugten Ergebnisses zu kennzeichnen. Unser Server befindet sich anschließend wieder im Zustand "Bereit".

2 Analyse der Produktfunktionen

In diesem Kapitel wird das Verhalten für die einzelnen Produktfunktionen analysiert. Dies geschieht, um später eine geeigneten Architektur realisieren zu können, auf Basis der im Pflichtenheft analysierten Produktfunktionen und nicht-funktionalen Anforderungen, die realisiert werden müssen.

2.1 Produktfunktionen

Hier werden die einzelnen Produktfunktionen des gesamten Systems analysiert. Dazu wird für jede Produktfunktion ein Sequenzendiagramm dargestellt sowie ein erläuternder Text bereitgestellt. Folgende Funktionsabläufe dienen als Gerüst für die tatsächliche Implementierung des Systems.

2.1.1 Analyse von Funktionalität <F1>: Auswertung einer ISAN

Die Funktion zur Auswertung einer ISAN arbeitet nach folgendem Ablauf:

Ein Externes-Smart-System (in unserem Fall ein Auto) sendet eine POST-Request (postRequest(ISAN)) über eine REST-API an den Web-Server und übermittelt eine ISAN, welche Informationen über den Unfall eines Autos enthält.
Der Web-Server analysiert die ISAN, indem er die Methode parseISAN(ISAN) aufruft. Diese Methode gibt die extrahierten Daten (result.data) zurück, sowie, ob die Extraktion erfolgreich war oder nicht (result.success), bzw. ob die ISAN erfolgreich analysiert werden konnte oder nicht.
Falls die Analyse erfolgreich war (result.success = True) erstellt der Web-Server einen neuen Auto-Unfall (createCarCrash(result.data)) mit den extrahierten Daten (result.data). Das erstellen des Auto-Unfalls (CarCrash) geschieht dabei asynchron, denn der Unfall hat keine im Programmablauf festgelegte Laufzeit. Das Löschen des Unfalls geschieht in der Funktion <F6>.
Im Anschluss gibt der Web-Server eine Erfolgsantwort (responseSuccess) für die Anfrage des Externen-Smart-Systems zurück.
Wenn die ISAN jedoch nicht im korrekten Format vorliegt, sendet der Web-Server eine Fehlerantwort (responseFailure) an das Externe-Smart-System zurück.

Der beschriebene Ablauf ist im folgendem Sequenzdiagramm dargestellt.

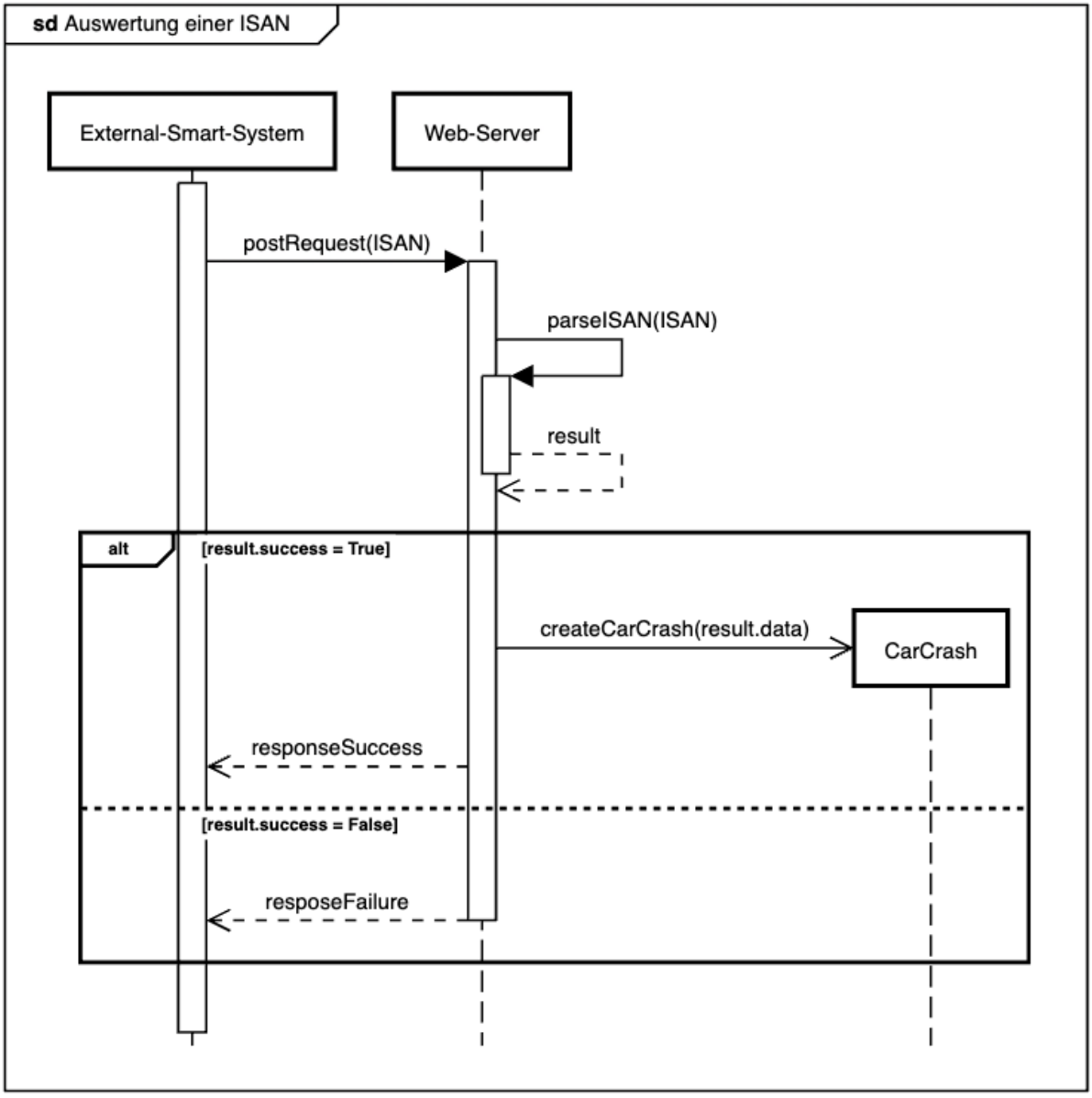

Abbildung 2.1: Auswertung einer ISAN (https://www.sequencediagram.org)

2.1.2 Analyse von Funktionalität <F2>: Zusammengehörende Unfälle werden gruppiert

Diese Funktion gruppiert die von den Fahrzeugen einzeln versendeten Unfallnachrichten und vom System bereits überprüften Nachrichten zu Unfällen mit allen beteiligten Fahrzeugen zusammen.
Die Funktion searchAndGroupAccident(carCrash) gruppiert auf dem Webserver die von der Funktion <F1> erstellten carCrashs zu einzelnen Accidents, die dann aus mehreren carCrashs bestehen. Die Unfälle werden an Hand ihres Zeitstempels (Zeitpunkt des Zustandekommens des Unfalls, nicht der Zeitpunkt des Nachrichteneingang im System) und ihrer Geodaten (GPS-Koordinaten des Unfallfahrzeuges) zu einem zusammengehörigen Unfall gruppiert.
Es kommt zu einer Gruppierung, wenn die Zeitstempel um nicht mehr als 1 Minuten abweichen und der Abstand der Unfallfahrzeuge nicht mehr als 100m beträgt.
Die zeitliche Differenz ist so bemessen, da davon auszugehen ist, dass die in den Fahrzeugen verbauten Sicherheits- und Rettungssysteme im Falle eines entsprechend schweren Unfalls unverzüglich ihre Nachrichten versenden und gleichzeitig noch z.B. ein verzögerter Auffahrunfall auf den Unfall mit aufgenommen wird.
Die Distanz zwischen den Fahrzeugen wird größer gewählt, da auch mögliche Unfallverläufe, wie z.B. das Herabrutschen eines Fahrzeuges entlang einer Böschung berücksichtigt werden müssen, die zur Entfernung der beteiligten Unfallfahrzeuge führen.
Um nun die neue Meldung (carCrash) eines Fahrzeuges einem Unfall zuzuordnen, überprüft die Funktion, ob bereits ein passender Unfall (Accidents) vorhanden ist, auf den die Guppierungsbedinungen passen. Für den Fall, dass bereits ein passender Unfall mit einer ID existiert, wird der carCrash dem Accident hinzugefügt (update(carCrash)), indem der hinzugefügte carCrash ebenfalls die ID des bereits existierenden Unfalls (Accident) erhält Für den Fall, dass kein passender Unfall gefunden wird, wird ein neuer Accident (NewAccident) erzeugt (createAccident(carCrash)). Zum Schluss returned die Funktion den accident an den Webserver. Die gesamte Kommunikation zwischen den Objekten erfolgt mittels synchroner Nachrichten, da auf Grund des geschachtelten pyramidenartigen Aufbaus erst eine weitere Bearbeitung stattfinden kann, wenn der vorherige Prozess abgeschlossen ist. z.B.: Erst, wenn carCrash einem Accident oder NewAccident zugeordnet wurde, soll die searchAndGroupAccidents(carCrash) beendet werden.

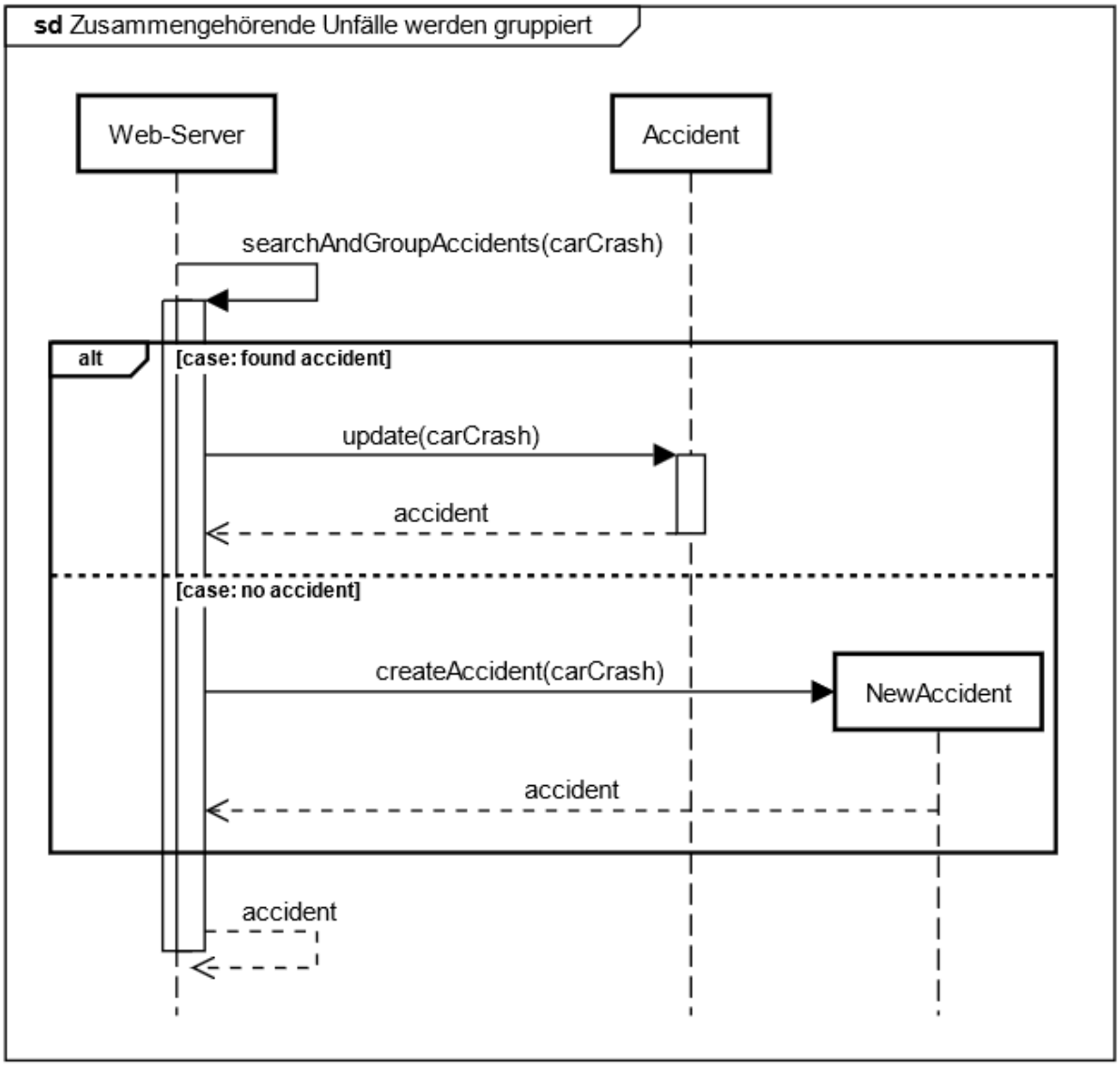

Abbildung 2.2: Zusammengehörende Unfälle werden gruppiert (https://www.sequencediagram.org)

2.1.3 Analyse von Funktionalität <F3>: Simulation eines Unfalls

Folgender Ablauf beschreibt, wie ein Unfall simuliert wird: Grundsätzlich hat diese Funktion zwei Hauptakteure, den Web-Server und den Simulations-Server. Der Web-Server kommuniziert noch mit einem CarCrash, welcher die nötigen Informationen enthält, um eine Simulation durchzuführen Der Simulations-Server selber kontrolliert dann intern noch zwei weitere Entitäten, den BeamNG-Simulator und ein Crash-Szenario.
Der BeamNG-Simulator ist eine Instanz der Physiks-Engine BeamNG.tech. Ein Crash-Szenario beschreibt einen Simulationsablauf auf der Physiks-Engine und behandelt dabei die komplette Simulation des Unfalls bis zum Sammeln und Auslesen der relevanten Daten.

Der Ablauf der Funktion beginnt damit, dass der Web-Server das CarCrash-Objekt nach den Daten fragt (getRelevantSimulationData()), welche für die Simulation benötigt werden, z.B. die Geschwindigkeit. Anschließend übermittelt der Web-Server dem Simulations-Server eine Get-Request (getRequest(data)) über die REST-API mit den erforderlichen Daten für die Simulation. Diese Anfrage passiert asynchron, d.h. der Web-Server ist danach völlig frei weiter nutzbar und wartet nicht auf eine Antwort des Simulations-Servers. Die Daten, welche dem Simulation-Server übermittelt werden sind dabei z.B. die Geschwindigkeit beim Unfall, die Anzahl der Insassen, etc.
Es können nun zwei Fälle auftreten:
1. Der Simulations-Server ist bereit eine Simulation durchzuführen (mehr dazu unten).
2. Der Simulations-Server ist gerade mit einer anderen Simulation beschäftigt und die Anfrage wird abgewiesen ("responseFailure")

Wenn der Simulations-Server bereit ist startet er die Simulation, indem er den Befehl start-Simulation(data) an den BeamNG-Simulator sendet und die Daten des Web-Servers einfach weitergibt.
Der BeamNG-Simulator erstellt dann ein Szenario (CrashScenario) für den Unfall (createScenario(data)), indem abermals die Daten des Web-Servers einfach weitergegeben werden. Dieses Szenario führt dann eine Unfallsimulation durch, welche mit der Methode simulateCrash() gestartet wird.
Nachdem die Unfallsimulation abgeschlossen ist, gibt das Szenario das Ergebnis der Simulation (simulationResult) an den BeamNG-Simulator zurück. Im Ergebnis der Simulation ist z.B. das 3D-Modell des Unfallautos enthalten. Das CrashScenario wird daraufhin nicht mehr benötigt und zerstört.
Der BeamNG-Simulator gibt das Simulationsergebnis (simulationResult) an den Simulations-Server zurück.
Schließlich sendet der Simulations-Server eine Erfolgsantwort an den Web-Server (responseSuccess). In dieser Antwort ist das Simulationsergebnis enthalten. Zuletzt wird das CarCrash-Objekt noch mit dem Ergebnis aus der Simulation gefüttert (update(responseSuccess.simulationResult)).

Das nachfolgende Sequenzdiagramm (2.6) zeigt den zuvor beschrieben Ablauf.

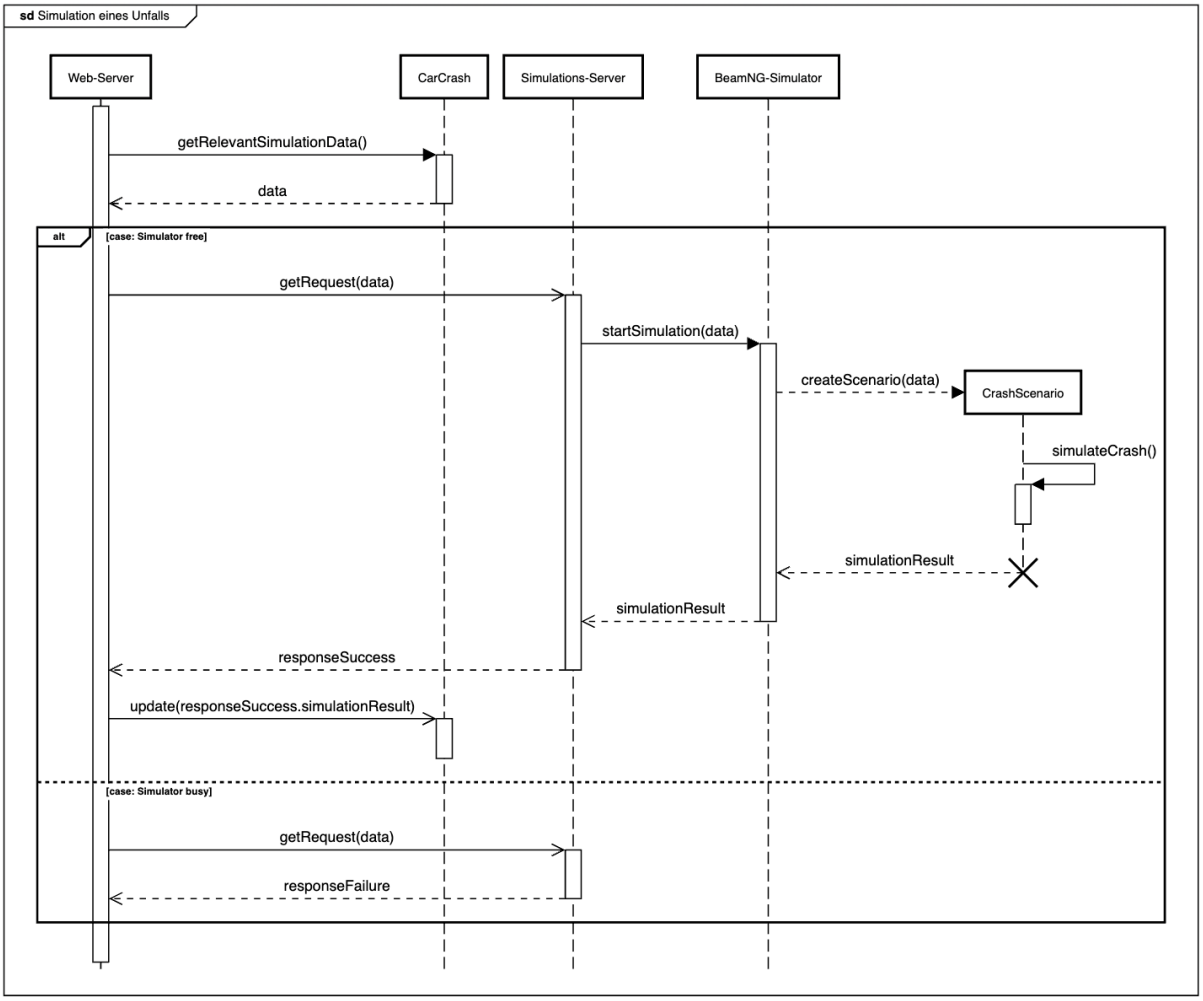

Abbildung 2.3: Simulation eines Unfalls (https://www.sequencediagram.org)

2.1.4 Analyse von Funktionalität <F4>: Visualisierung eines Unfalls auf der Karte

Die Funktion sorgt dafür, dass der Unfall auf der Karte an entsprechender Geoposition und in 3D-Darstellung visualisiert wird.
Es wird damit gestartet, dass die Website das CarCrash-Objekt nach den verfügbaren Daten fragt (getRelevantVisualizationData()), welche für das Visualisieren der simulierten 3D-Modelle der Unfallfahrzeuge benötigt werden. Die Website erhält als return diese Daten (data). Die Anfrage erfolgt synchron, da erst mit der Visualisierung begonnen werden kann, wenn die Daten vorliegen.
Hat die Website die Daten erhalten, übermittelt die Website mit Hilfe der Funktion visualizeAccidents(coordinates, 3D-Model) die benötigten Daten (Koordinaten des jeweiligen Fahrzeuges, sowie das simulierte 3D-Modell) für die Visualisierung auf der Karte an die Google Maps Api.
Die Google Maps Api rendert dann das Overlay für die Karte auf der Website mit den 3D-Modellen an den entsprechenden Koordinaten.

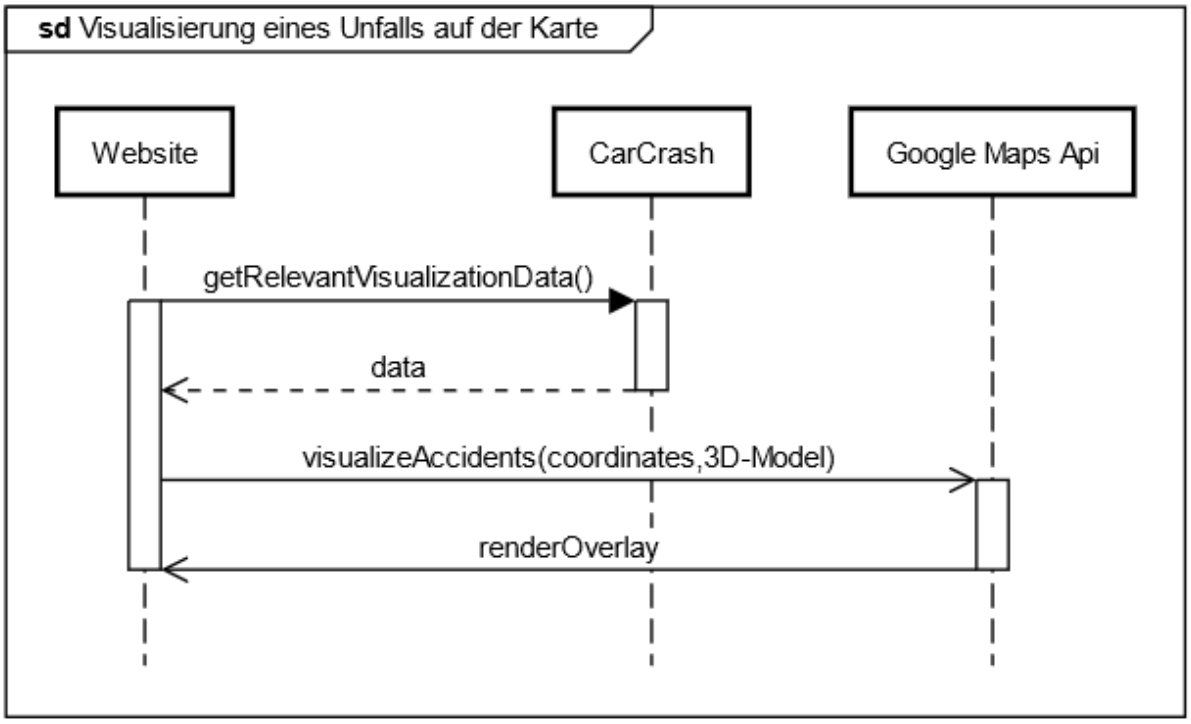

Abbildung 2.4: Visualisierung eines Unfalls auf der Karte (https://www.sequencediagram.org)

2.1.5 Analyse von Funktionalität <F5>: Suchen eines Unfalls

Die Funktion ermöglicht es dem User einen entsprechenden Unfall zu suchen.
Die ID eines Unfalls wird von dem Benutzer in das Suchfeld auf der Startseite der Webanwendung eingegeben (enter accidentID) und mit der Auswahl des Suchen-Buttons gestartet.
Beim Eingeben der Daten updatet die Website die Suchleiste und zeigt die eingegebenen Ziffern an (update searchbar). Dies wird solange gemacht bis der User auf den Suchenknopf (search button) klickt.
Anschließend liefert die Funktion get(accidentID) dem Webserver die accidentID um nach dem entsprechenden Unfall suchen zulassen. Dies geschieht mit Hilfe von searchAccidents(accidentID), die im Erfolgsfall die zur gesuchten ID gehörenden Daten als accident zurückliefert, welche widerum an die Website zurückgegeben werden
Findet sich kein passender Eintrag mit der entsprechenden ID liefert die Funktion ein false zurück und auf der Webseite erscheint ein Hinweis, dass kein passender Eintrag gefunden wurde.

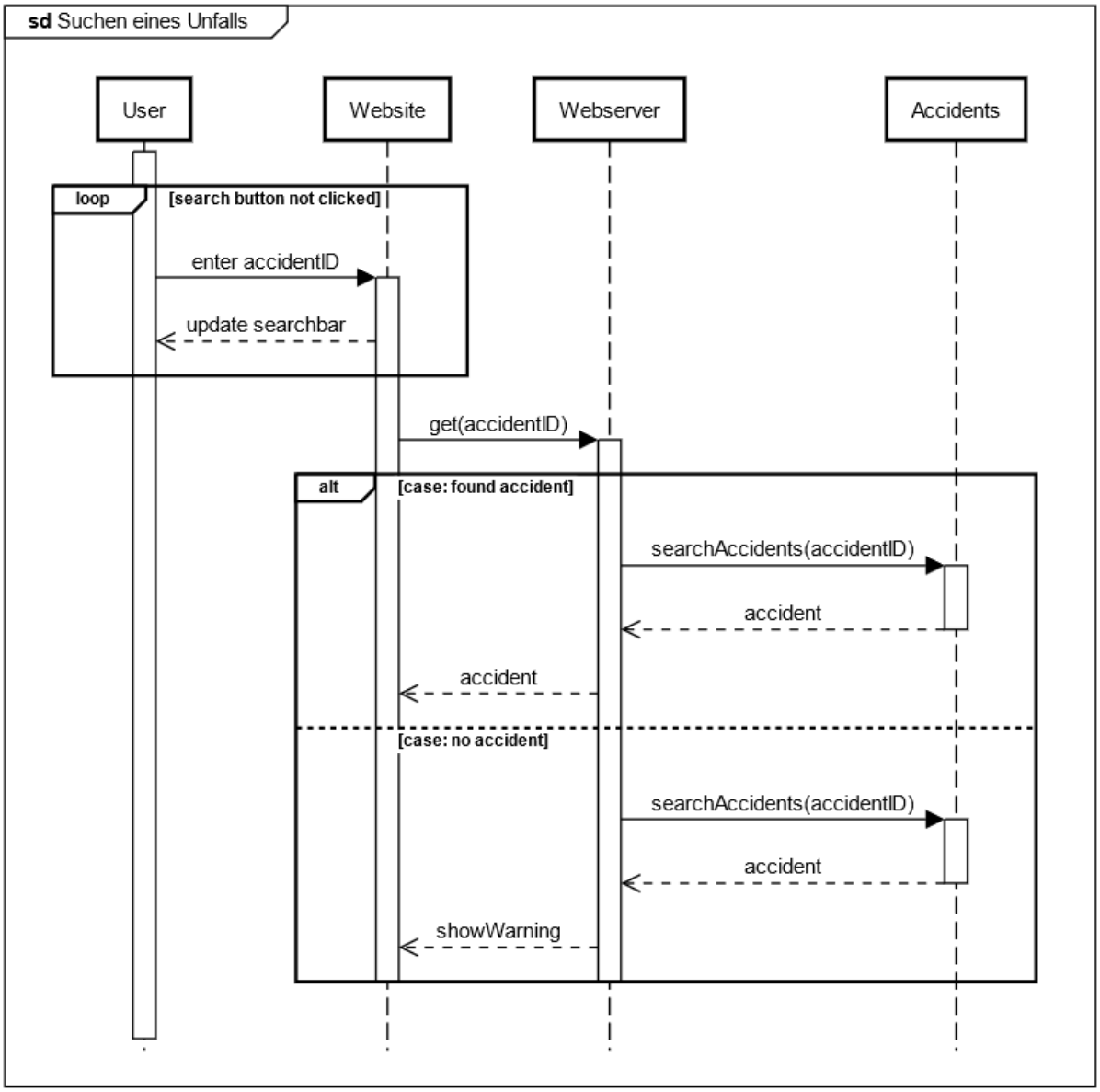

Abbildung 2.5: Suchen eines Unfalls (https://www.sequencediagram.org)

2.1.6 Analyse von Funktionalität <F6>: Löschen eines Unfalls

Im Folgenden wird der Prozess beschrieben, wie ein Unfall gelöscht wird. Dies ist einer der Hauptzwecke des Web-Servers, weshalb in der nachfolgenden Beschreibung auch detailliert seine Funktionsweise erläutert wird.

Der Web-Server löst mittels der Methode checkDelete() das Überprüfen abgelaufener Unfälle aus.
Daraufhin geht der Web-Server in eine Schleife über, welche für jeden vorhandenen Unfall (Accident) die Zeit der Erstellung abfragt (getTimeOfCreation()). Diese wird ihm dann übermittelt (timeOfCreation).
Daraufhin überpüft der Web-Server, ob der jeweilige Unfall schon länger als ein vordefieniertes Zeitlimit existiert (Accident exists longer than timelimit). Ist dies der Fall, wird der entsprechende Unfall gelöscht (deleteAccident()).

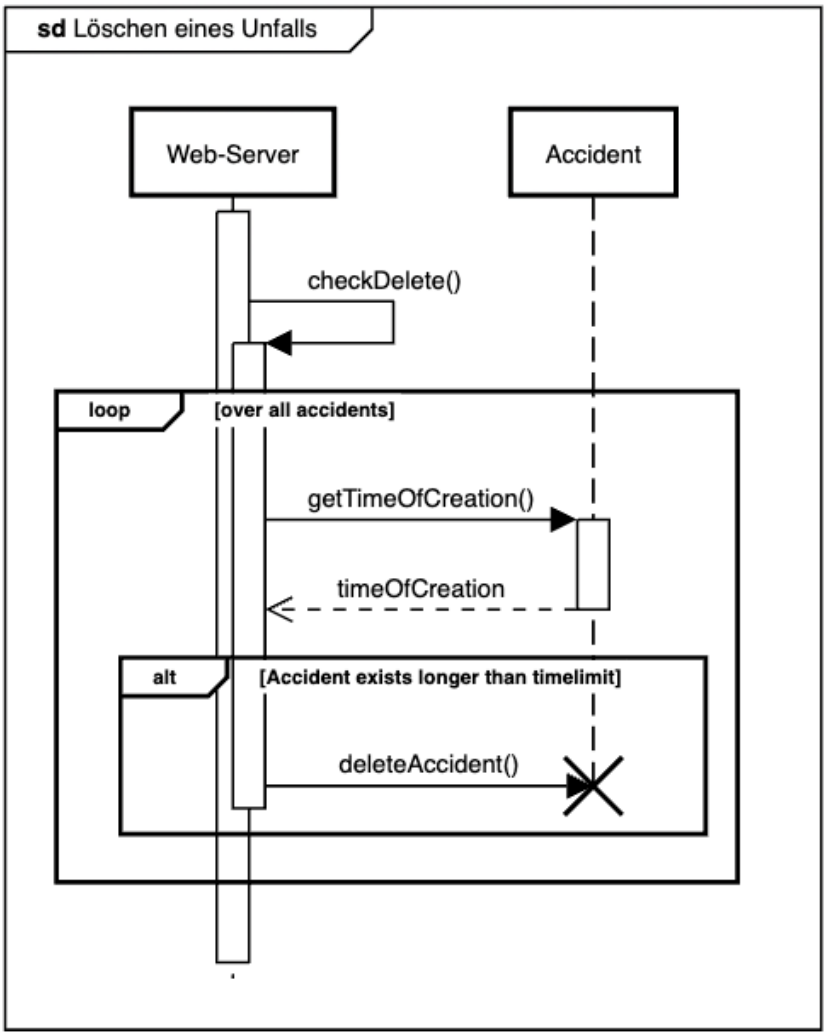

Abbildung 2.6: Löschen eines Unfalls (https://www.sequencediagram.org)

2.1.7 Analyse von Funktionalität <F7>: Aufrufen der Rettungskarte

Die Funktion zum Aufruf der Rettungskarte funktioniert wie folgt: Der Nutzer möchte die Rettungskarte eines bestimmten Autos öffnen und wählt den entsprechenden Link aus (openRettungskarte(car)). Daraufhin fordert die Website aus dem Unfall die Rettungskarte des ausgewählten Autos an (getRettungskarte(car)) und kriegt den Link dazu zurückgeliefert (RettungskarteLink).
Schlussendlich öffnet die Website dann diesen Link (openRettungskarte(RettungskarteLink)) in einem neuen Tab.

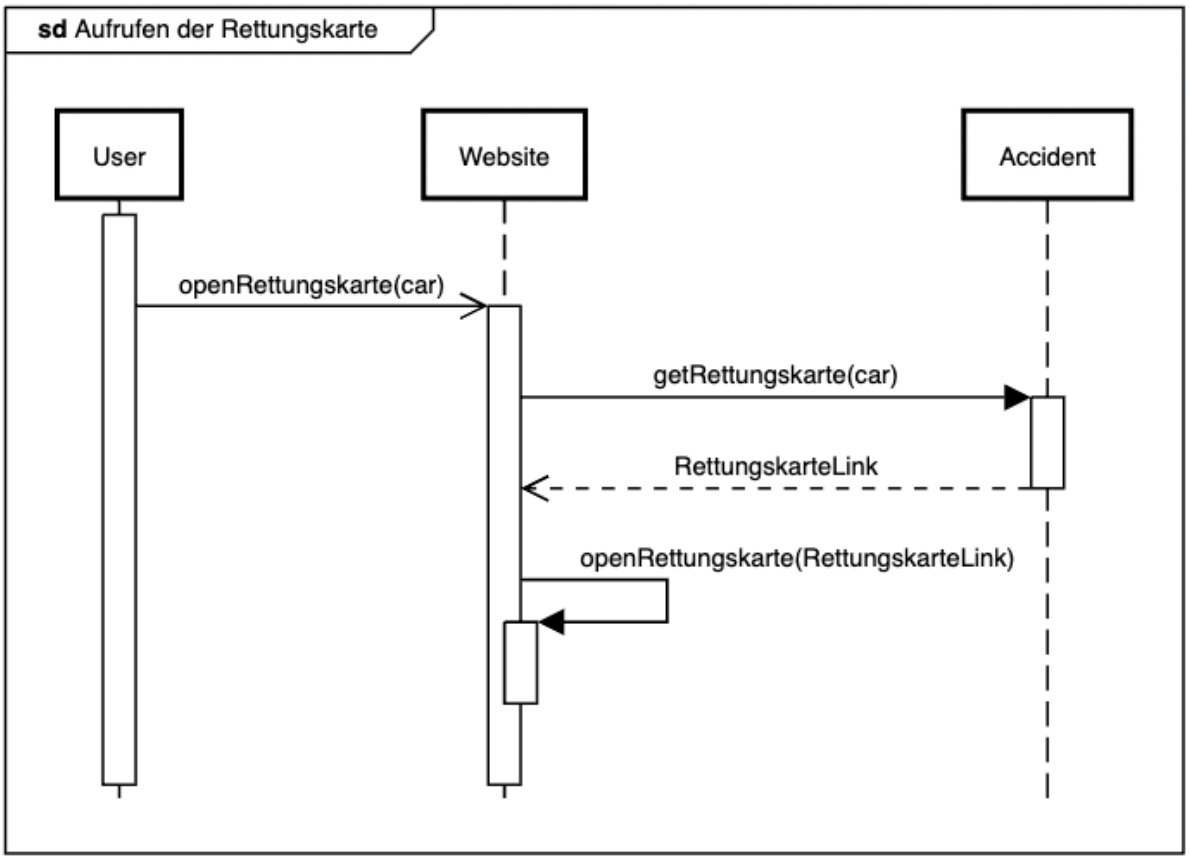

Abbildung 2.7: Aufrufen der Rettungskarte (https://www.sequencediagram.org)

2.1.8 Analyse von Funktionalität <F8>: Anzeigen der Fahrzeuginformationen

Die Funktion zur Anzeige der Fahrzeuginformationen funktioniert nach dem im Folgenden beschriebenen Schema. Der Nutzer möchte die zusätzlichen Informationen eines bestimmten Autos in einem separaten Fenster öffnen und fährt mit dem Mauszeiger über das entsprechende Auto(hoverOver(car)). Daraufhin fordert die Website aus dem Unfall die zusätzlichen Informationen zu dem ausgewählten Auto an (getInformation(car)) und kriegt diese zurückgeliefert (information).
Schlussendlich öffnet die Website dann ein separates Fenster mit diesen Informationen (displayInformationWindow(information)).

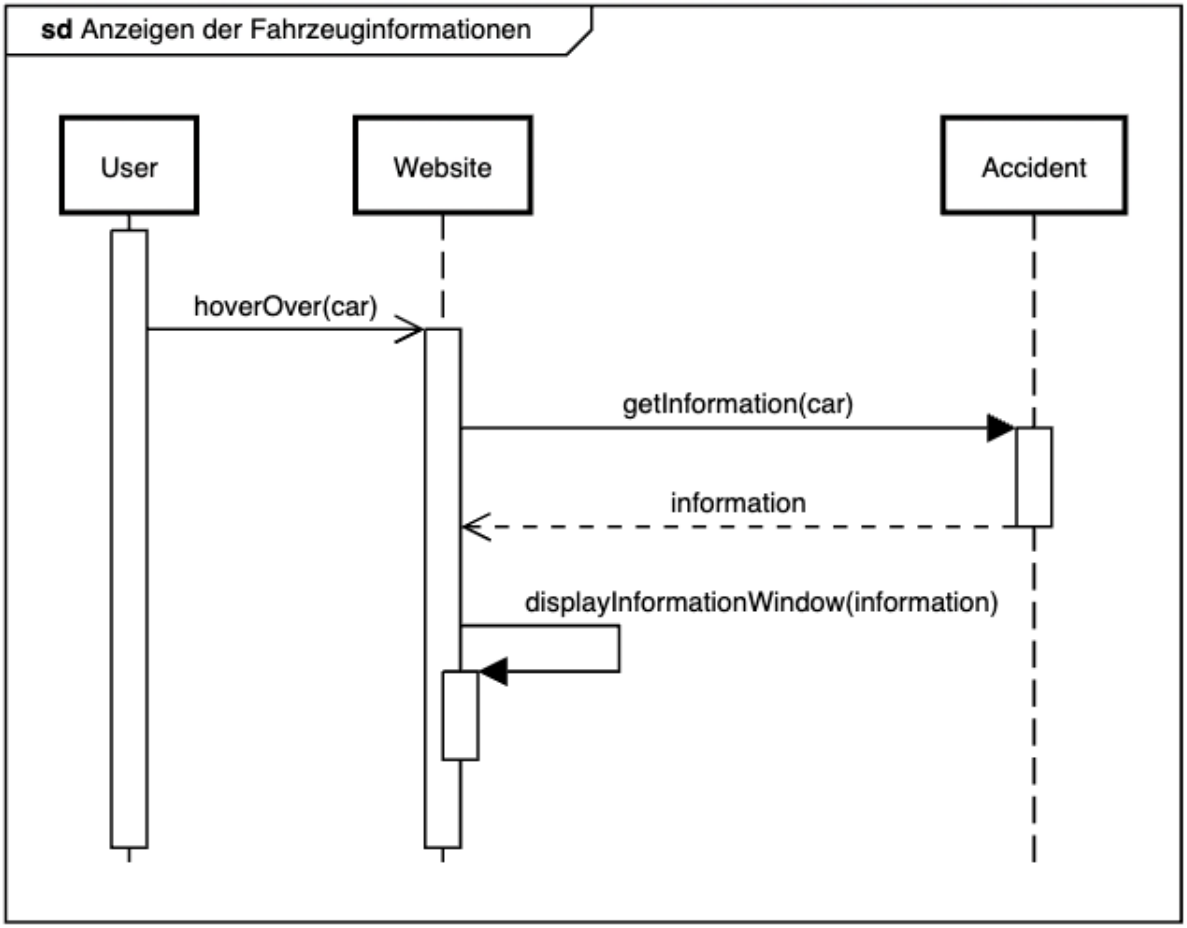

Abbildung 2.8: Anzeige der Fahrzeuginformationen (https://www.sequencediagram.org)

2.1.9 Analyse von Funktionalität <F9>: Anzeige der Krafteinwirkung auf Insassen

In dieser Funktion soll ein Fenster geöffnet werden, welches die Krafteinwirkung auf die Insassen des Autos anzeigt, wenn der Nutzer ein Auto auswählt.
Der Nutzer wählt also auf der Website ein Auto aus (selectCar(car)).
Daraufhin fragt die Website das ausgewählte Auto (CarCrash) nach den Krafteinwirkungen auf die Insassen (getOccupantsForce()), welche es zurückgibt (occupantsForce). Daraufhin kann die Website dann das Fenster mit diesen Informationen anzeigen (showForceInformationWindow()).
Zu irgendeinem Zeitpunkt wählt der Nutzer das Auto wieder ab und damit wird das Fenster auch wieder geschlossen.

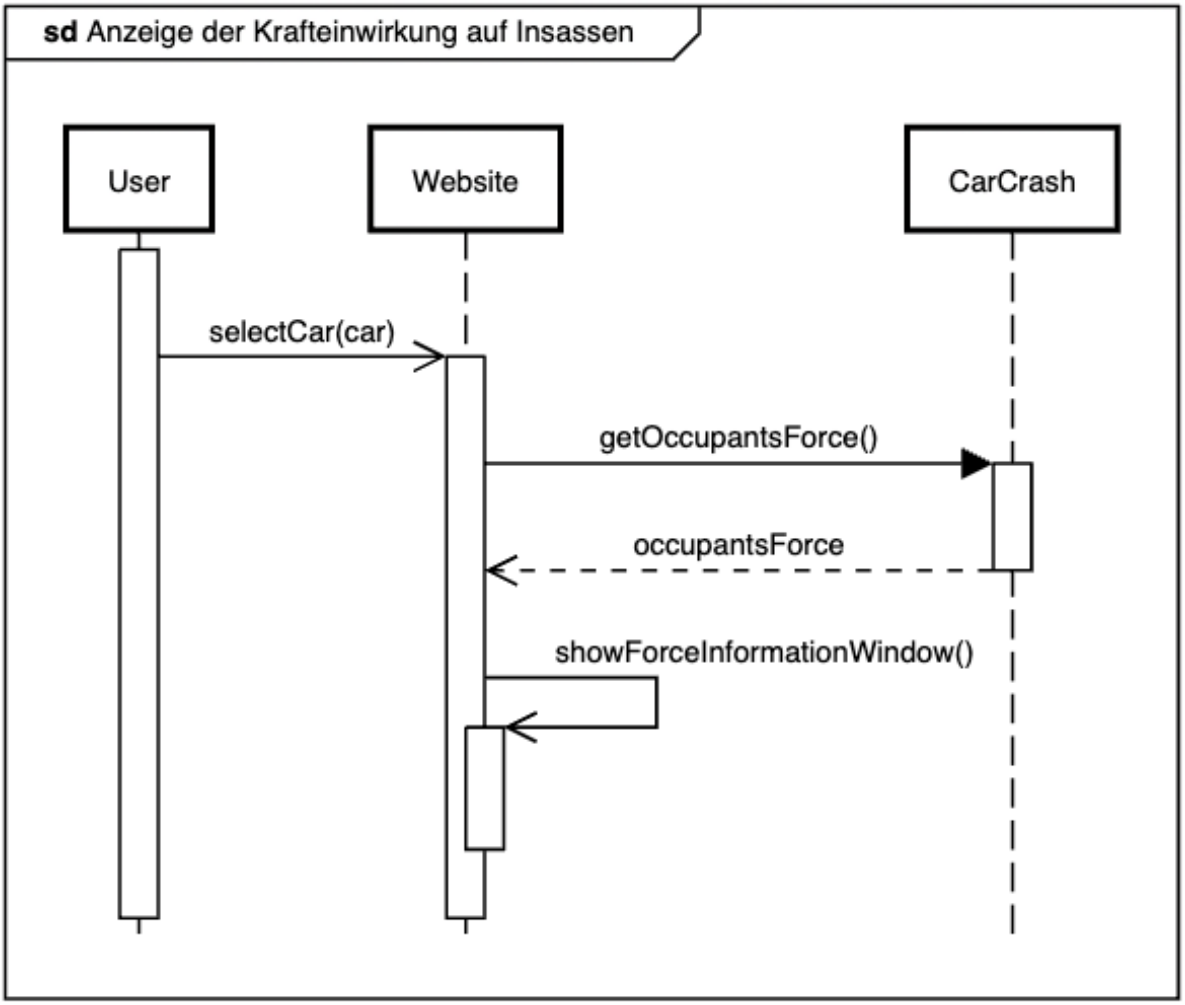

Abbildung 2.9: Anzeige der Krafteinwirkung auf Insassen (https://www.sequencediagram.org)

2.2 Nichtfunktionale Anforderungen

In diesem Anschnitt werden die nicht-funktionalen Anforderungen näher erläutert. Dazu werden für die jeweiligen Aspekte Tabellen erstellt, welche die Relevanz bei der Entwicklung angeben. Die Nichtfunktionalen Anforderungen, welche bereits im Pflichtenheft aufgeführt wurden, wurden nicht verändert und wie beschrieben umgesetzt.

2.2.1 Funktionalität

Produktqualität	**sehr gut**	**gut**	**normal**	**nicht relevant**
Angemessenheit	x			
Richtigkeit	x			
Interoperabilität	x			
Ordnungsmäßgkeit				x

Angemessenheit : Es ist erforderlich, dass die Anwendung alle vorgegebenen Funktionen erfüllt.
Richtigkeit : Richtigkeit ist wichtig zu betrachten, da sie direkt Einfluss auf die Notfallreaktion hat und die Fähigkeit der Einsatzkräfte beeinflusst, genaue Informationen korrekt abzurufen.
Interoperabilität : Es ist unerlässlich, dass unsere Anwendung in der Lage ist, die ISAN-Nummer eines Fahrzeugs abzurufen. Darüber hinaus ist es wichtig, dass der Client die Informationen und simulierten Mechanismen vom Server empfangen kann. Aus diesem Grund ist die Interoperabilität ein entscheidendes Element unserer Anwendung.
Ordnungsmäßgkeit : Unser Produkt wird nicht als endgültiges Produkt entwickelt. Datenschutz und andere Faktoren müssen zu einem späteren Zeitpunkt berücksichtigt werden

2.2.2 Sicherheit

Produktqualität	**sehr gut**	**gut**	**normal**	**nicht relevant**
Zuverlässigkeit	x			
Reife		x		
Fehlertoleranz		x		
Wiederherstellbarkeit		x		

Zuverlässigkeit : Die Webanwendung wird von Einsatzkräften genutzt, um den Ort eines Unfalls zu lokalisieren und relevante Daten wie Verletzungen und Rettungskarten einzusehen. Es ist wichtig, dass unsere Software diese Funktionen zuverlässig ausführen kann
Reife : Die Reife der Software bedeutet, dass weniger Fehler auftreten können. Dies ist von entscheidender Bedeutung, da ein Fehler zu einem fehlgeschlagenen Einsatz führen kann.
Fehlertoleranz : Es ist wichtig, Fehler zu vermeiden. Allerdings sollte unsere Software auch

dann korrekt arbeiten, wenn zum Beispiel die Verarbeitung eines Unfalls fehlschlägt. In diesem Fall sollte die Karte weiterhin alle anderen Unfälle fehlerfrei anzeigen.
Wiederherstellbarkeit : Wenn die Webanwendung nicht in der Lage ist, neue Daten vom Server zu laden, muss sie die vorhandenen Daten beibehalten können. Es ist wichtig, dass die Webanwendung auch in diesem Fall ordnungsgemäß funktioniert.

2.2.3 Benutzbarkeit

Produktqualität	**sehr gut**	**gut**	**normal**	**nicht relevant**
Verständlichkeit	x			
Erlernbarkeit			x	
Bedienbarkeit		x		
Effizienz	x			
Zeitverhalten		x		
Verbrauchsverhalten				x

Verständlichkeit : Die Informationen und Daten müssen für die Einsatzkräfte klar und verständlich sein, um eine effektive Bewältigung der Situation zu gewährleisten
Erlernbarkeit : Die Webanwendung ist nicht für den normalen Nutzer gedacht, sondern soll von geschulten Einsatzkräften bedient werden.
Bedienbarkeit : Die Webanwendung muss benutzerfreundlich gestaltet sein, um Verzögerungen bei der Bedienung zu vermeiden.
Effizienz : Da die Webanwendung gelegentlich viele 3D-Objekte auf einer Karte anzeigen muss, ist es wichtig, dass sie effizient programmiert wird, um Verzögerungen oder langsame Software zu vermeiden.
Zeitverhalten : Die Webanwendung muss in Echtzeit Unfalldaten verarbeiten und diese schnell auf der Benutzeroberfläche anzeigen, um eine schnelle Reaktion zu ermöglichen.
Verbrauchsverhalten : Die Webanwendung wird auf einem geeigneten PC laufen und die Ressourcen des PCs können vollständig genutzt werden. Daher ist das Verbrauchsverhalten als weniger relevant anzusehen.

2.2.4 Änderbarkeit

Produktqualität	sehr gut	gut	normal	nicht relevant
Analysierbarkeit		x		
Modifizierbarkeit		x		
Stabilität	x			
Prüfbarkeit		x		
Übertragbarkeit				x
Anpassbarkeit		x		
Installierbarkeit				x
Konformität			x	
Austauschbarkeit		x		

Analysierbarkeit und Prüfbarkeit : Eine gewisse Prüfbarkeit und Analysefähigkeit sind wichtig, um sicherzustellen, dass die Anwendung korrekt funktioniert, die notwendigen Funktionen erfüllt und Fehler schnell identifiziert und behoben werden können.
Modifizierbarkeit, Austauschbarkeit und Anpassbarkeit : Die Software sollte erweiterbar sein, um zukünftige Funktionen hinzuzufügen oder vorhandene zu ersetzen. Zum Beispiel könnten spezielle Einsatzdienste oder neue Daten, die von zukünftigen Autos bereitgestellt werden, geeignete Funktionen erfordern. Daher muss die Software austauschbar, modifizierbar und anpassbar sein.
Stabilität : Die Webanwendung müssen stabil arbeiten, um die Daten korrekt und zeitnah liefern zu können.
Übertragbarkeit und Installierbarkeit : Die Software muss auf den geeigneten Geräten, auf denen sie getestet wurde, einwandfrei funktionieren. Es besteht keine Notwendigkeit, dass sie auf vielen verschiedenen Plattformen arbeitet.
Konformität : Die Software muss auf verschiedenen unterstützten Geräten und Browsern konsistente Ergebnisse liefern, um eine hohe Konformität sicherzustellen.

2.2.5 Qualitätsanforderungen

Die als am wichtigsten eingestuften Qualitätsmerkmale werden nun operationalisiert, indem detaillierte Produktanforderungen festgelegt werden und die einzuhaltenden Richtlinien (z.B. Standards und Normen) angegeben werden

- $\langle Q10 \rangle$ Die Funktion <F3> sollte benutzerfreundlich gestaltet und einfach zu bedienen sein.
- $\langle Q20 \rangle$ Die Sprache der Webanwendung soll deutsch sein
- $\langle Q30 \rangle$ Die Informationen auf der Webanwendung sollen verständlich sein .
- $\langle Q40 \rangle$ Das Produkt soll auf üblichen Browsern laufen.
- $\langle Q50 \rangle$ Das Produkt soll in der Lage sein, 3D-Objekte auf einer Karte anzuzeigen, ohne dass die Systemleistung beeinträchtigt wird.
- $\langle Q60 \rangle$ Das Produkt soll stabil laufen.
- $\langle Q70 \rangle$ Die Korrektheit der Datenübertragung vom Server zum Client soll überprüft werden.
- $\langle Q80 \rangle$ Der Code soll ausreichend kommentiert und klar modularisiert sein.
- $\langle Q90 \rangle$ Die Übertragung des Unfalls auf die Karte sollte innerhalb von 3 Sekunden erfolgen.

3 Resultierende Softwarearchitektur

Dieses Kapital befasst sich mit der Systemarchitektur. Es beschreibt die verschiedenen Komponenten, aus denen das System besteht, und erläutert die Beziehung zwischen diesen Komponenten.

3.1 Komponentenspezifikation

Die verschiedenen Komponenten werden in einem übersichtlichen Komponentendiagramm visualisiert, das eine visuelle Darstellung der Architektur des Systems bietet. Jede Komponente wird dabei beschrieben und ihre Rolle im Gesamtsystem werden erläutert.

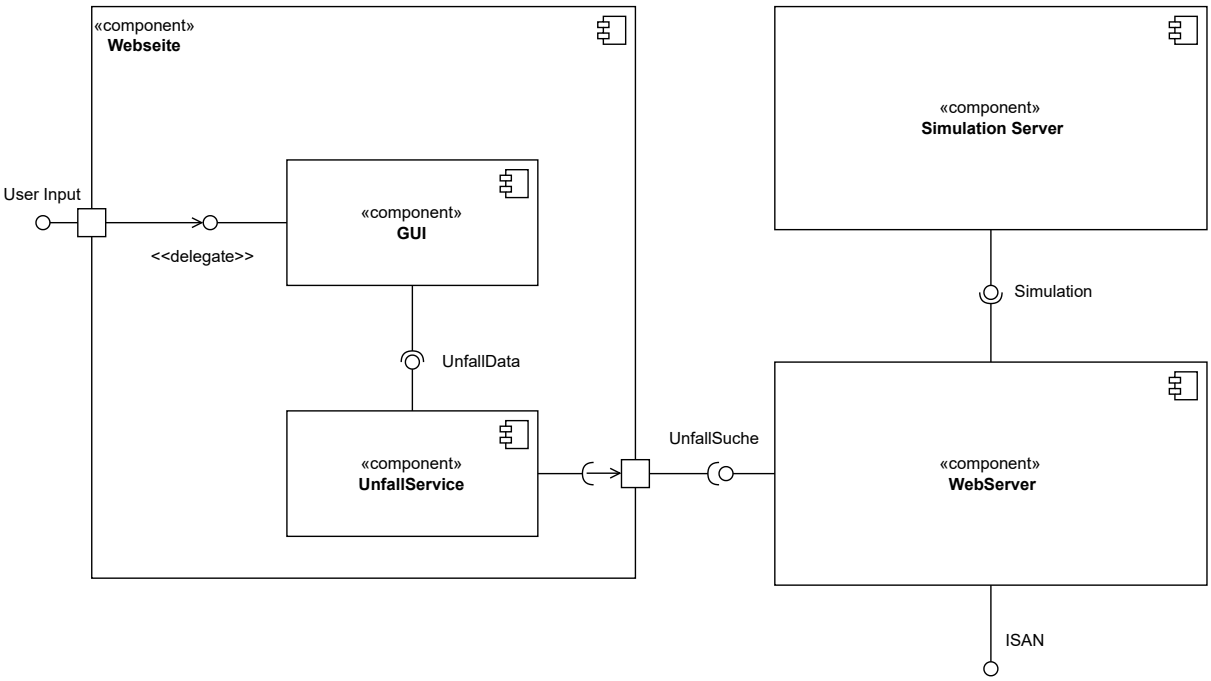

Abbildung 3.1: Komponentendiagramm.

Komponente $\langle C10 \rangle$: GUI
Die GUI-Komponente ermöglicht es dem Benutzer, eine Unfall-ID einzugeben. Anschließend werden die Informationen zu dem Unfall angezeigt und es wird die Möglichkeit geboten, interaktiv auf eine Rettungskarte zuzugreifen, diese zu vergrößern und zu verschieben.

Komponente $\langle C20 \rangle$: UnfallService
Die unfallService-Komponente ermöglicht es dem GUI, eine Unfall-ID einzugeben und aus dieser ID den zugehörigen Unfall vom Webserver abzurufen. Anschließend wird der Unfall im Speicher gespeichert, um später ohne ID darauf zugreifen zu können.

Komponente $\langle C30 \rangle$: Webserver
Die Webserver-Komponente ermöglicht das Hinzufügen eines Unfalls anhand einer ISAN (International Standard Accident Number). Anschließend wird eine Simulation dieses Unfalls vom Simulationsserver angefordert und der Unfall lokal gespeichert. Zudem ermöglicht die Komponente die Suche nach einem Unfall anhand der zugehörigen Unfall-ID im UnfallService.

Komponente $\langle C40 \rangle$: Simulations-Server
Die Simulationsserver-Komponente bietet die Möglichkeit, einen Unfall durch die Eingabe der Unfallparameter zu simulieren und anschließend die resultierenden Daten zurückzugeben.

3.2 Schnittstellenspezifikation

In diesem Abschnitt werden die Schnittstellen, die im Komponentendiagramm dargestellt sind, beschrieben und ihre Operationen erläutert.

Schnittstelle $\langle I10 \rangle$: UnfallData

Operation	**Beschreibung**
Accident getAccidentById (Id)	Wenn ein Unfall mit dieser ID existiert, wird dieser zurückgegeben. andernfalls wird null zurückgegeben.
Accident getAccident ()	Es wird der zuletzt gespeicherte Unfall zurückgegeben, was es der GUI-Komponente ermöglicht, auf den Unfall zuzugreifen, ohne jedes Mal die ID angeben zu müssen.

Schnittstelle $\langle I20 \rangle$: UnfallSuche

Operation	**Beschreibung**
Accident serachAccidentById (Id)	Es wird im lokalen Speicher nach einem Unfall mit der angegebenen ID gesucht. Falls ein Unfall mit der ID existiert, werden die entsprechenden Daten zurückgegeben. Andernfalls wird eine Fehlermeldung ausgegeben.

Schnittstelle $\langle I30 \rangle$: ISAN

Operation	**Beschreibung**
void addAccident (ISAN)	Die Daten werden aus der ISAN extrahiert und entweder ein neuer Unfall erstellt oder die Daten werden einem bestehenden Unfall zugeordnet.

Schnittstelle $\langle I40 \rangle$: Simulation

Operation	**Beschreibung**
SimulationData simulateCrash (speed, force, hsn ,tsn)	Es wird eine Simulation eines Unfalls mit den eingegebenen Parametern durchgeführt und das Ergebnis der Simulation wird zurückgegeben.

3.3 Protokolle für die Benutzung der Komponenten

3.3.1 GUI

Die Ausführung des Programms ist mithilfe eines Web-browser Fenster. Die Webseite muss in einem Browser geöffnet werden. Nach dem Öffnen zeigt das Programm das Unfall-ID Suchfeld. Ein korrekte Unfall-ID soll dann eingegeben werden und danach wird die Unfall Informationsbildschirm angezeigt. Dort wird die Informationen vom Unfall z.B. die Karte, Verletzungen der Passagiere, Automodelle angezeigt. Die Rettungskarte kann mit einem Klick angezeigt und die Karte kann auch vergrößert bzw. verkleinert sein. Mit einem Klick auf die Home-Taste wird die Unfall-ID Suchfeld wieder angezeigt. Das Programm wird beendet wenn das Webbrowser-Fenster geschlossen ist.

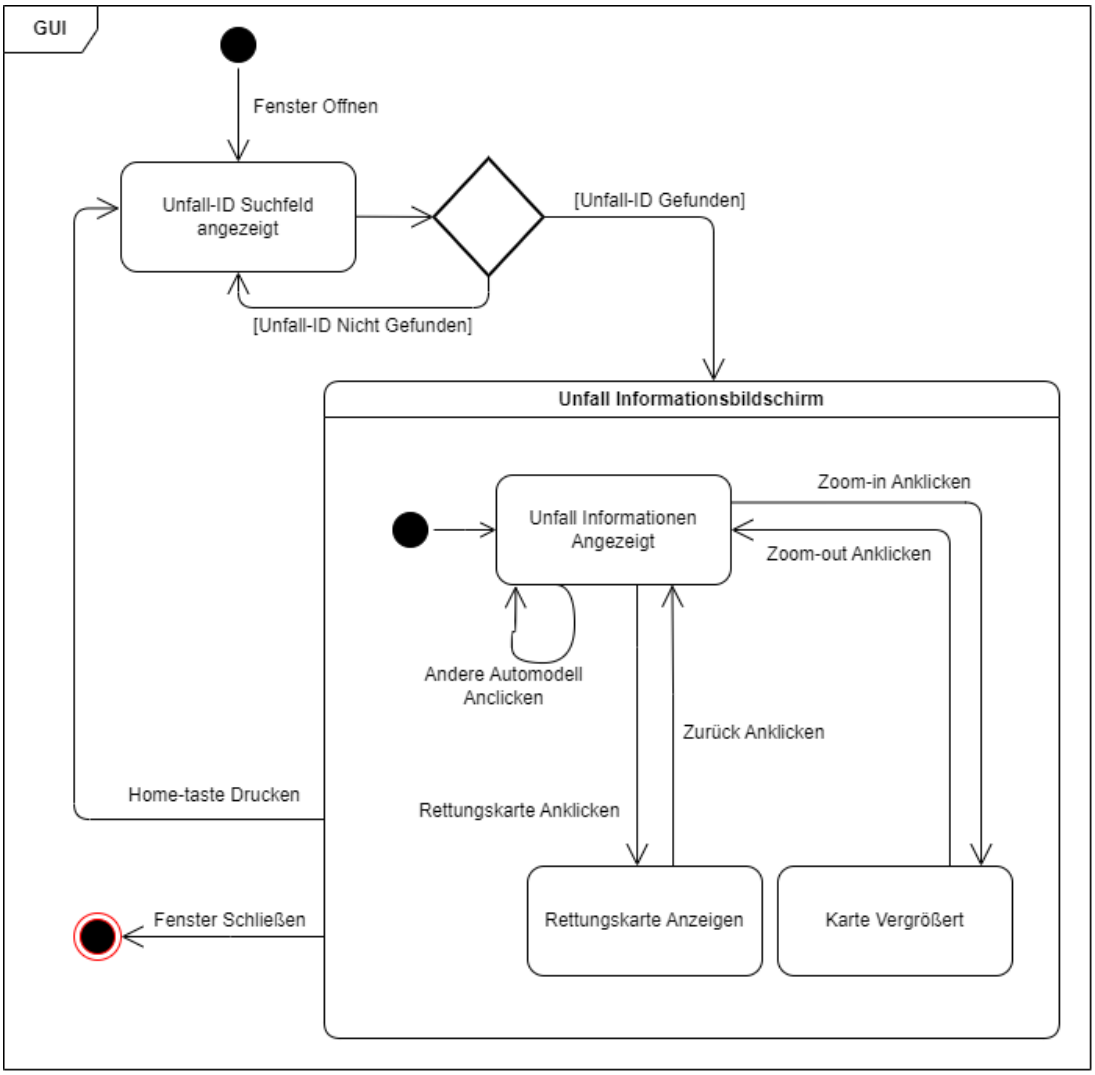

Abbildung 3.2: State-Chart der GUI

3.3.2 UnfallService

Die UnfallService wird gestartet und befindet sich auf dem Zustand bereit. Falls es einen Unfall-ID erhalten, wird es dann im Webserver geprüft ob es im lokalen Speicher existiert. Falls es gefunden wird, wird den Unfall Abgerufen und im Speicher gespeichert, damit es ohne ID zugegriffen werden kann. Danach befindet sich der UnfallService wieder in Bereit Zustand.

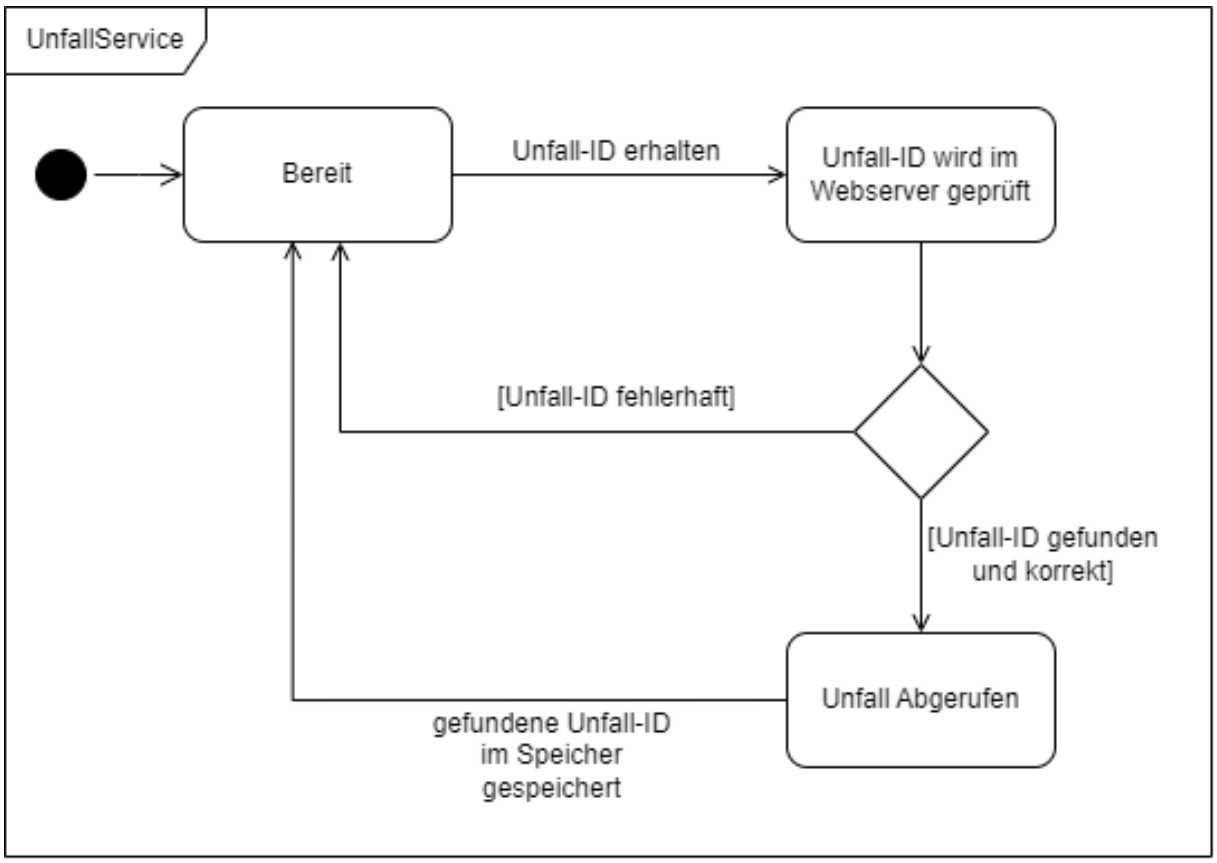

Abbildung 3.3: State-Chart des Unfallservices

3.3.3 Webserver

Zunächst wird das Webserver gestartet und befindet sich auf Bereit Zustand. Falls es ein ISAN erhalten, wird die Webserver die Unfall auf dem lokalen Server hinzugefügt und eine Simulation wird vom Webserver angefordert. Die Webserver bekommt dann die Informationen von den simulierten Unfällen und dies wird dann lokal gespeichert. Die gespeicherte Unfällen ist nun dann auf dem Webserver verfügbar und kann mithilfe des Unfall-IDs zugegriffen werden. Am Ende ist das Webserver wieder auf dem Zustand bereit.

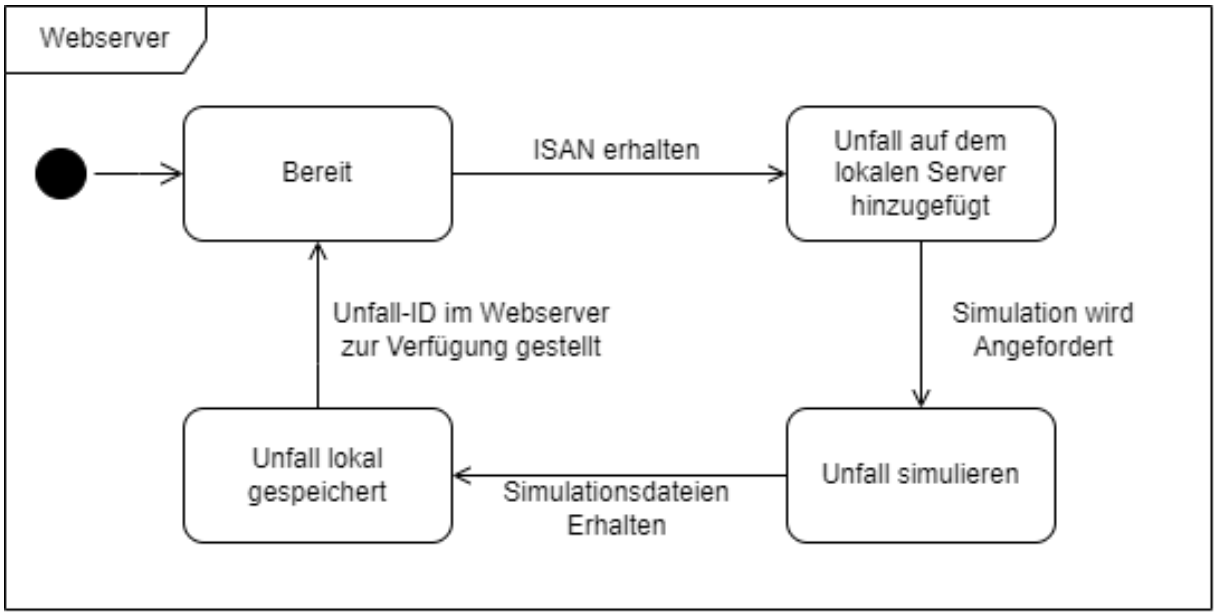

Abbildung 3.4: State-Chart des Webservers

3.3.4 Simulationsserver

Die Simulationsserver wird gestartet und befindet sich auf dem Zustand bereit. Falls es eine Simulationsanfrage mit dem benötigten Parameter erhalten, wird dann die Simulation von dem Unfall durchgeführt. Nach dem die Simulation fertig ist, gibt es die Dateien zurück und am Ende befindet sich wieder das Simulationsserver auf dem Zustand bereit.

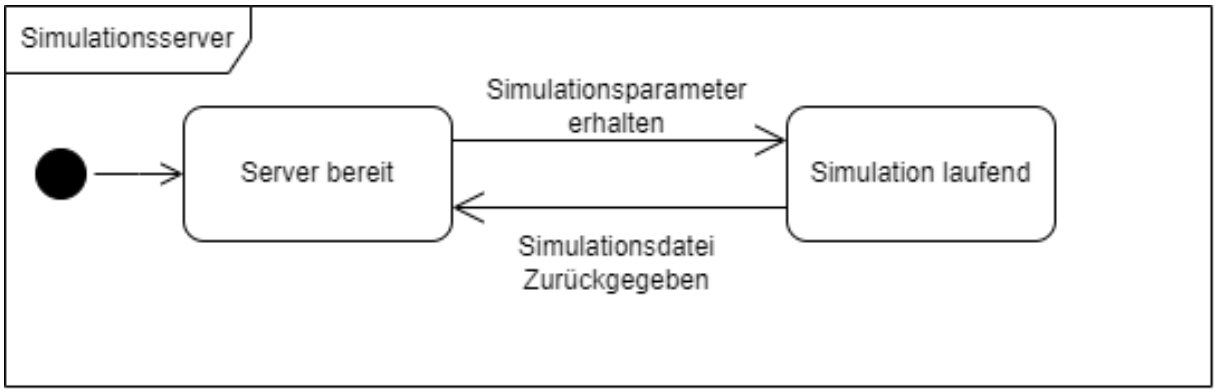

Abbildung 3.5: State-Chart der Simulationsserver

4 Verteilungsentwurf

In diesem Kapitel erfolgt eine Beschreibung der Verteilung unseres Systems.

Unser System setzt sich aus drei verschiedenen Geräten zusammen. Das erste ist das Client-Gerät, auf dem unsere Webseite in einem Webbrowser ausgeführt wird. Hier haben wir die Möglichkeit, beliebig viele Client-Geräte über HTTP-Anfragen mit dem Webserver-Gerät zu verbinden. Das Client-Gerät ist mit Windows 10 ausgestattet und verfügt über eine Node.js-Umgebung, in der unsere Webserver-Komponente aktiv ist.
Der Webserver-Gerät wiederum steht in Verbindung mit einem Simulationsserver-Gerät, das ebenfalls über HTTP erreichbar ist. Auf dem Simulationsserver-Gerät ist Windows 10 installiert und es ist eine Python 3-Umgebung vorhanden. Hier laufen unsere Simulationsserver-Komponenten.
Die Kommunikation zwischen dem Webserver-Gerät und dem Simulationsserver-Gerät erfolgt über die entsprechenden HTTP-Anfragen und -Antworten. Durch diese Verbindung wird eine reibungslose Datenübertragung und Interaktion zwischen den beiden Geräten gewährleistet. Somit ermöglicht unser System eine effektive Zusammenarbeit zwischen den Clients, dem Webserver und dem Simulationsserver, um eine umfassende Funktionalität und eine realistische Simulationserfahrung zu bieten.

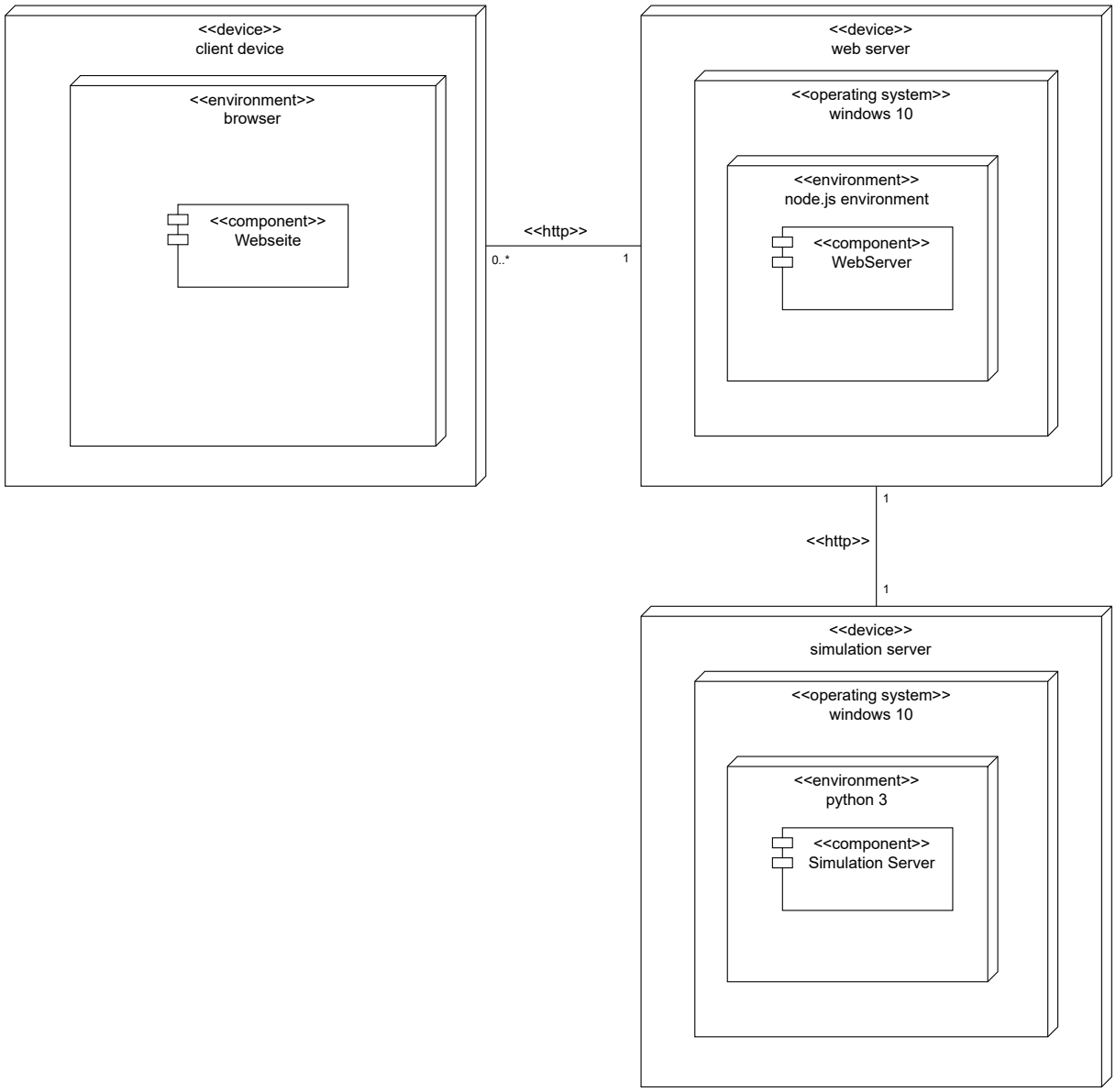

Abbildung 4.1: Verteilungsdiagramm

5 Implementierungsentwurf

Dieser Abschnitt hat die Aufgabe, alle verwendeten Klassen und Bibliotheken zu dokumentieren. Dabei wird jede Komponente aus Kapitel 3 gesondert betrachtet. Für Entwurfsentscheidungen, die mehr als eine Komponente betreffen, wird mit Verweisen zwischen den Dokumentationen der Komponente gearbeitet. Es sind dabei so viele Unterabschnitte einzufügen, wie Komponenten vorhanden sind. Die Anzahl sollte also mit den Komponenten aus Kapitel 3 übereinstimmen.

5.1 Implementierung von Komponente <C10>: GUI

Die GUI-Komponente ermöglicht es dem Benutzer, eine Unfall-ID einzugeben. Anschließend werden die Informationen zu dem Unfall angezeigt. Auf einer Karte sollen die simulierten Fahrzeugmodelle angezeigt werden. Desweiteren kann für jedes entsprechende Unfallfahrzeug auf eine Rettungskarte zu gegriffen werden. Auf Grund von Einrichtungsschwierigkeiten des Google-Kontos, welches für die Erstellung der 3D-Visualisierung mittels der Google Maps JavaScript API zwingend benötigt wird, konnten die Funktionen, die die Karte betreffen noch nicht programmiert werden. Wie besprochen werden im nach folgenden die aktuell existierenden Bereiche dargestellt.

5.1.1 Paket-/Klassendiagramm

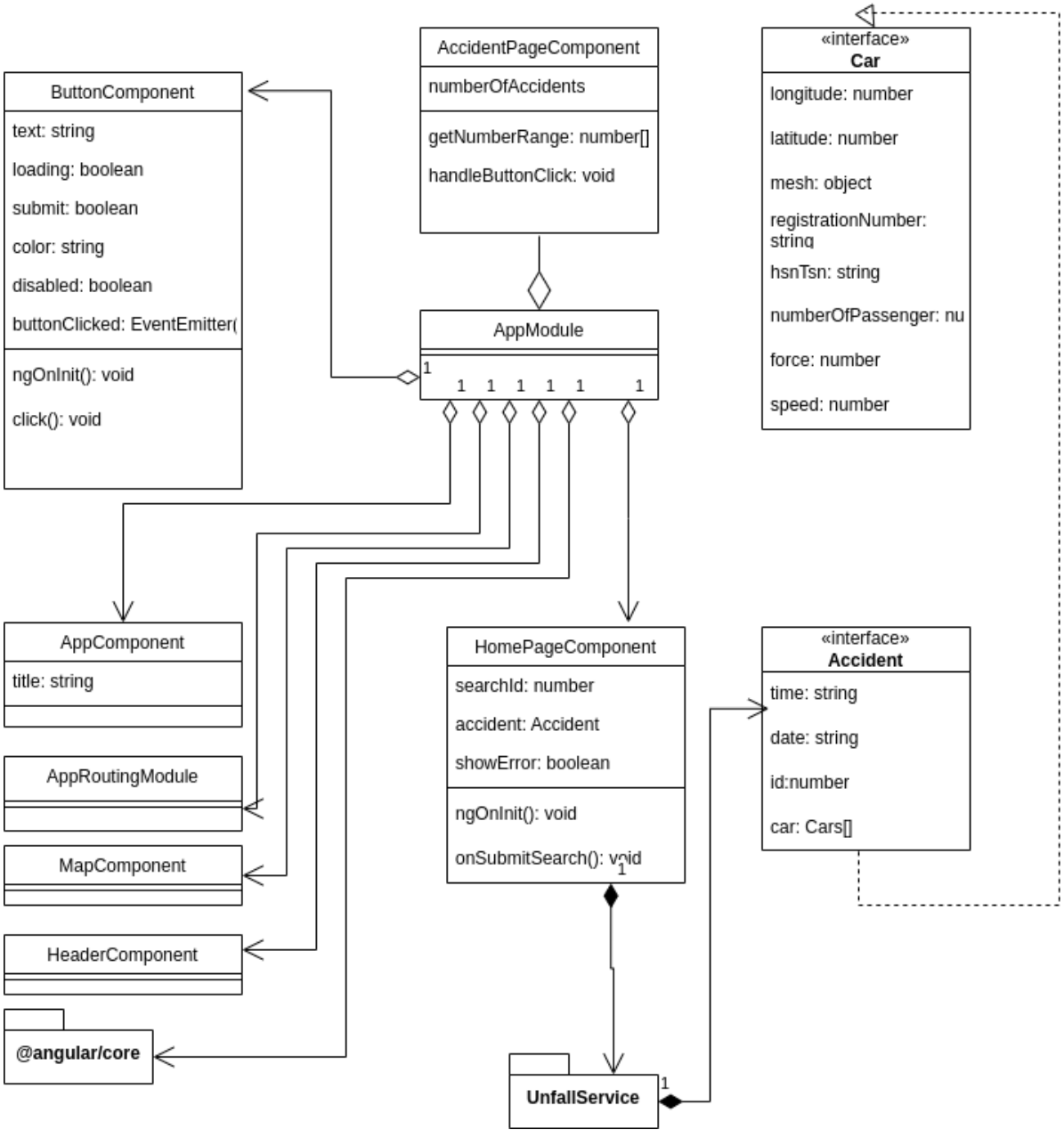

Abbildung 5.1: Klassendiagramm für Komponente $\langle C10 \rangle$ (erstellt mit draw.io)

5.1.2 Erläuterung

ButtonComponent⟨$CL10$⟩

Aufgabe

Erstellen und Auswertung eines Buttons

Attribute

text: string Text der im definierten Button steht loading: boolean submit: boolean color: string Farbe des Button disabled: boolean buttonClicked: EventEmitter()

Operationen

ngOnInit() bei Initialisierung der Anwendung wird diese ausgeführt und der button erstellt. click() Lauschen, ob der Button ausgelöst wurde

Kommunikationspartner

Wird eingebunden in AppModul

AccidentPageComponent⟨$CL11$⟩

Aufgabe

Anzeigen der Seite mit allen Informationen des Unfalls nach erfolgreicher Eingabe einer Unfall-ID

Attribute

numberOfAccidents: number

Operationen

getNumberRange() handleButtonClick()

Kommunikationspartner

Wird eingebunden in AppModul

AppComponent⟨$CL12$⟩

Aufgabe

Bereitstellung der Grundinformationen

Attribute

title: string Namen der Anwendung festlegen

Kommunikationspartner

Wird eingebunden in AppModul und stellt die Grundkomponente bereit für AppModul

AppRoutingModul⟨$CL13$⟩

Aufgabe

Bereitstellung des Routings zwischen den Seiten der Website

Attribute

keine

Kommunikationspartner

Wird eingebunden in AppModul

MapComponent⟨$CL14$⟩

Aufgabe

Visualisierung der 3D-Modelle auf der Karte

Attribute

-wird noch programmiert

Kommunikationspartner

Wird eingebunden in AppModul

HeaderComponent⟨$CL15$⟩

Aufgabe

Bereitstellung des Headers mit Überschrift und Logo des PLRIs

Attribute

keine

Kommunikationspartner

Wird eingebunden in AppModul

@angular/core⟨$CL16$⟩

Aufgabe

Bereitstellung der vom Framework Angualr mit gelieferten Core-Elemente

Attribute

keine

Kommunikationspartner

Wird eingebunden in AppModul und steht allen anderen Modulen auch zur Verfügung

HomePageComponent⟨$CL17$⟩

Aufgabe

Nimmt die eingegebene Unfall_ID und sucht den passenden Unfall mit Hilfe des Unfall-Service

Attribute

searchid: number die eingegebene UnfallID
accident: Accident der Unfall der gefunden wurde
showError: boolean ob ein passender Unfall zur ID gefunden wurde oder nicht

Operationen

ngOnInit() Initialisierung bzw. reset des Unfallservice, wenn die Suchstartseite geladen wird

onSubmitSearch() den Unfallservice mit der Suche nach einem passenden Unfall mit der eingegebenen ID beauftragen

Kommunikationspartner

Wird eingebunden in AppModul und stellt den passenden Unfall mit Hilfe des Unfallservice bereit.

AppModul⟨$CL18$⟩

Aufgabe

Vereinen alle Funktionen, sodass die Website angezeigt werden kann

Attribute

keine

Kommunikationspartner

ButtonComponent, AppComponent, AppRoutingModul, MapComponent, @angular/core, HomepageComponent, AccidentComponent

Unfallservice⟨$CL19$⟩

siehe Implementierung von Komponente <C20>: UnfallService

car⟨$CL20$⟩

Aufgabe

Speicherung der Informationen jeweils eines unfallautos

Attribute

longtitude: number Längengrad der Autoposition latitude: number Breitengrad der Autoposition mesh: object gerendertes 3D Mesh des Autos registrationNumber: string Nummernschild hsnTsn: string HSNund TSN Nummer numberOfPassengers: number Anzahl der Insassen force: number Krafteinwirkung aufs Fahrzeug im Unfallmoment speed: number Geschwindigkeit des Fahrzeuges im Unfallmoment

Kommunikationspartner

Wird implementiert von Accident

Accident$\langle CL21 \rangle$

Aufgabe

Speicherung der Informationen jeweils eines Unfalls mit mehreren Fahrzeugen, gruppiert mit gleicher ID

Attribute

time: string Uhrzeit des Unfalls date: string Datum des Unfalls id: ID des Unfalls, alle beteiligten Fahrzeuge besitzen gleiche ID car:Car[] Modelle der Fahrzeuge

Kommunikationspartner

Wird genutzt von dem Unfallservice.

5.2 Implementierung von Komponente <C20>: UnfallService

Die UnfallService-Komponente ermöglicht es der GUI, eine Unfall-ID einzugeben und den entsprechenden Unfall vom Webserver abzurufen. Nach dem Abrufen des Unfalls wird er im Speicher gespeichert, um später ohne die Verwendung der ID darauf zugreifen zu können.

5.2.1 Paket-/Klassendiagramm

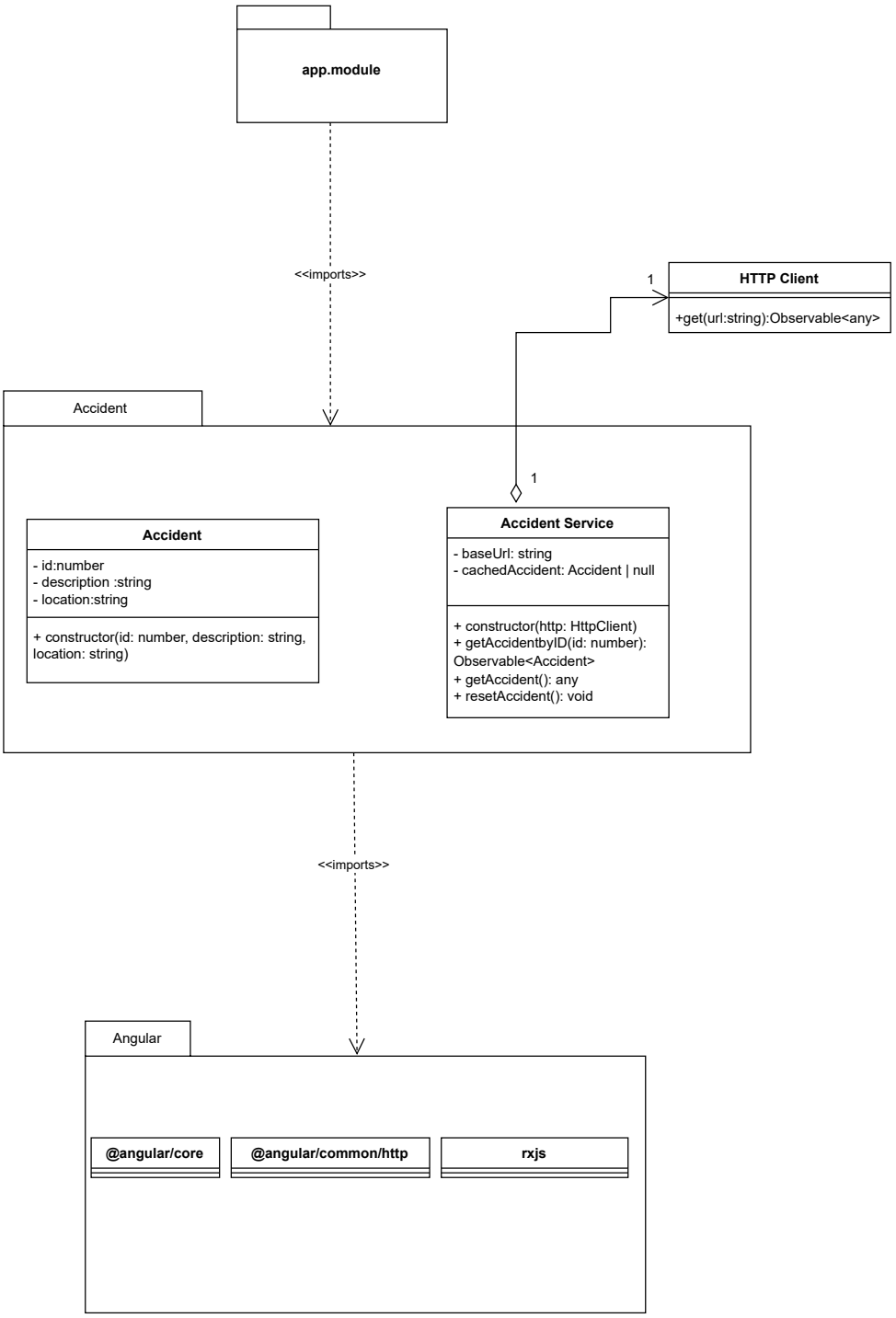

Abbildung 5.2: Klassendiagramm für Komponente $\langle C20 \rangle$ (erstellt mit draw.io)

Accident Service⟨$CL20$⟩

Aufgabe

Die Klasse Accident Service repräsentiert einen Service, der sich mit Unfall-bezogenen Operationen befasst.

Attribute

baseUrl (String): Eine Zeichenkette, die die Basis-URL für die Unfall-API enthält. cachedAccident (Accident | null): Eine Variable zur Zwischenspeicherung des abgerufenen Unfalls. Sie kann entweder ein Unfallobjekt (Accident) oder null sein.

Operationen

Konstruktor(http: HTTP Client): Eine Methode zum Erstellen einer neuen Instanz der Klasse Accident Service. Sie nimmt eine Instanz des HTTP Client entgegen, um HTTP-Anfragen durchführen zu können.

getAccidentbyID(id: number): Observable<Accident>: Eine Methode, die eine Unfall-ID entgegennimmt und ein Observable zurückgibt, das den abgerufenen Unfall repräsentiert. Sie verwendet den HTTP Client, um eine HTTP-Anfrage an den Webserver zu senden und den Unfall basierend auf der ID abzurufen. Die abgerufenen Daten werden zwischengespeichert und können über das Observable abonniert werden.

getAccident(): any: Eine Methode, die den zwischengespeicherten Unfall zurückgibt. Sie ermöglicht den Zugriff auf den Unfall, ohne eine ID anzugeben.

resetAccident(): void: Eine Methode, die den zwischengespeicherten Unfall zurücksetzt, indem sie die cachedAccident-Variable auf null setzt.

Kommunikationspartner

HTTP Client: Die Klasse Accident Service kommuniziert mit der HTTP Client-Klasse, um HTTP-Anfragen an den Webserver zu senden und Unfalldaten abzurufen.

Accident⟨$CL21$⟩

Aufgabe

Die Klasse Accident repräsentiert einen Unfall mit seinen zugehörigen Daten.

Attribute

id (number): Eine eindeutige ID, die den Unfall identifiziert. description (string): Eine Zeichenkette, die eine Beschreibung des Unfalls enthält. location (string): Eine Zeichenkette, die den Ort des Unfalls angibt.

Operationen

Konstruktor(id: number, description: string, location: string): Eine Methode zum Erstellen einer neuen Instanz der Klasse Accident. Sie nimmt die ID, die Beschreibung und den

Ort des Unfalls als Parameter entgegen und initialisiert die entsprechenden Attribute der Klasse.

Kommunikationspartner

Keine direkten Kommunikationspartner.

5.3 Implementierung von Komponente <C30>: Webserver

5.3.1 Teil 1: Webserver

Die Webserver-Komponente ermöglicht das Hinzufügen eines Unfalls durch Verwendung einer ISAN (Internationale Standard-Unfallnummer). Danach wird eine Anfrage an den Simulationsserver gestellt, um eine Simulation dieses Unfalls zu erhalten, die dann lokal gespeichert wird. Darüber hinaus ermöglicht diese Komponente die Suche nach einem Unfall anhand der zugehörigen Unfall-ID im UnfallService.

5.3.2 Teil 2: Routes

/accidents (POST):

Der Endpoint /accidents ermöglicht die Erstellung neuer Unfälle. Bei einer POST-Anfrage wird die Funktion ‘createIsan‘ aufgerufen. Diese Funktion prüft die International Standard Accident Number (ISAN), bevor sie den Unfall erstellt. Die Unfalldaten werden als JSON-Datei in der Datei accident$_{accidentId}$.$jsongespeichert$.

accidents/delete (DELETE):

Um unseren Unfallspeicher auf dem neuesten Stand zu halten, kümmert sich der Endpoint /accidents/delete um die Löschung abgelaufener Unfälle. Er verwendet eine periodische Anfragefunktion und prüft die gespeicherten Unfälle auf ihre Zeit und ihr Datum. Alle Unfälle, die die 5 Stunden überschritten haben, was auf einen Ablauf hinweist, werden automatisch gelöscht.

accidents/:id (GET):

Der Endpoint /accidents/:id erleichtert das effiziente Abrufen bestimmter Unfälle auf der Grundlage ihrer eindeutigen ID. Bei einer GET-Anfrage sucht der Server nach der entsprechenden JSON-Datei, accident_accidentId.json. Wird sie gefunden, wird der Inhalt der Datei gelesen und als JSON-Response mit den Unfalldetails zurückgegeben. Wird kein passender Unfall gefunden, wird eine entsprechende Antwort zurückgegeben, die angibt, dass es keinen Unfall mit der angegebenen ID gibt.

Diese Routen und Endpunkte ermöglichen zusammen ein umfassendes Unfallmanagementsystem, das die Erstellung von Unfällen, die regelmäßige Bereinigung und den effizienten Abruf auf der Grundlage eindeutiger IDs umfasst

5.3.3 Paket-/Klassendiagramm

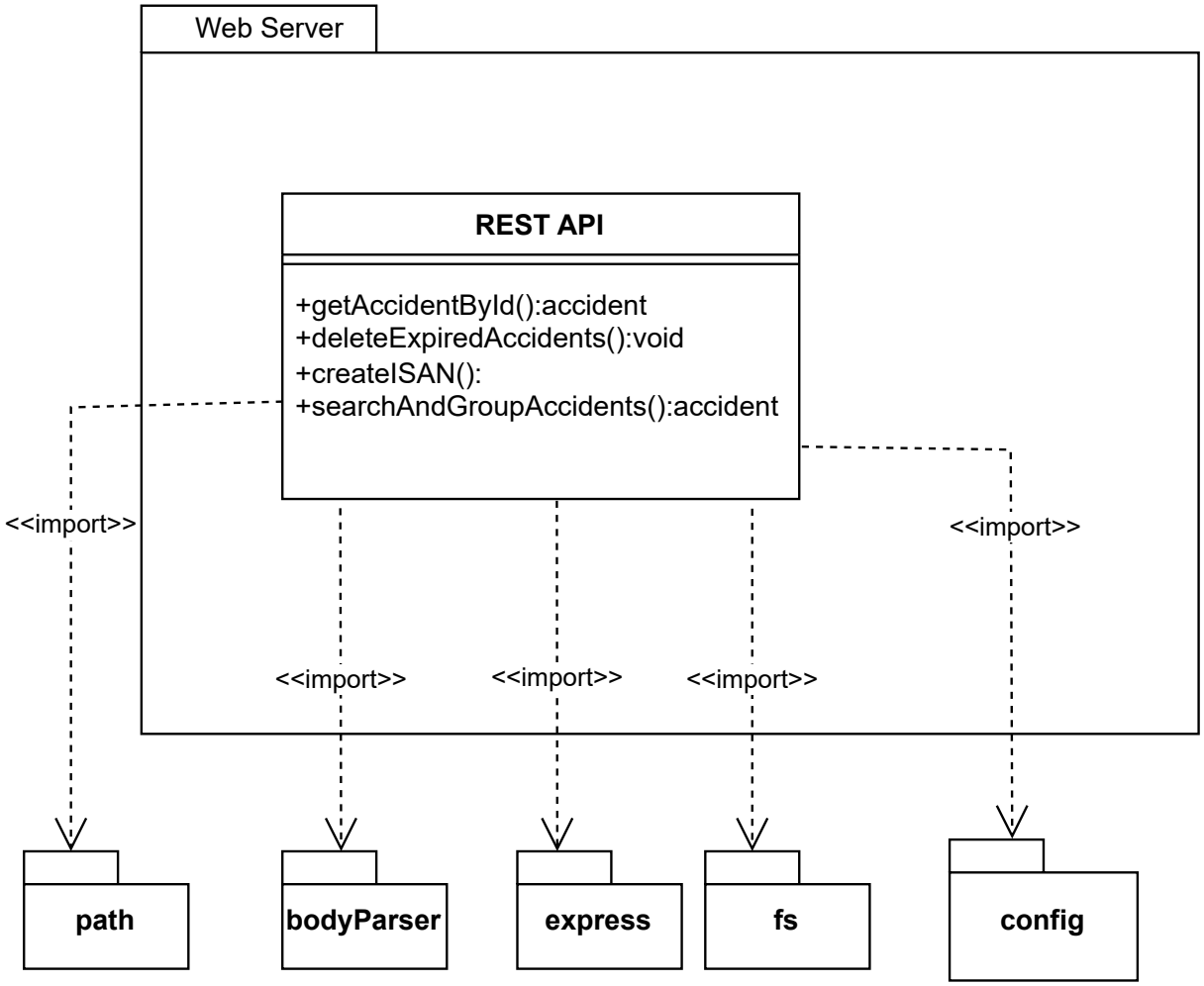

Abbildung 5.3: Klassendiagramm für Komponente $\langle C30 \rangle$ (erstellt mit draw.io)

5.3.4 Erläuterung

REST API⟨$CL30$⟩

Aufgabe

Umgang mit Unfällen, Erstellung, Wiederherstellung und Löschung

Attribute

Keine

Operationen

createISAN():Nimmt die Unfalldaten in der Anfrage auf, prüft die ISAN und erstellt den Unfallbericht. deleteExpiredAccidents():void:Nach dem Aufruf erfolgt ein regelmäßiger Aufruf, um Unfälle zu löschen, die vor mehr als 5 Stunden angelegt wurden.

getAccidentById():accident:Nimmt eine Unfall-ID als Anfrageparameter, sucht im Unfallordner nach der passenden Datei (accident_accidentId.json) und gibt den Unfall in der Datei zurück

searchAndGroupAccidents():accident: Durchsucht die Unfälle im Unfallordner nach Ähnlichkeiten in Entfernung und Zeitpunkt. Unfälle mit gleichem Ort und gleicher Zeit innerhalb eines bestimmten Schwellenwerts werden zu einem Unfall zusammengefügt, wobei die Autos der verschiedenen Unfälle mit einbezogen werden

Kommunikationspartner

Keine direkten Kommunikationspartner.

5.4 Implementierung von Komponente <C40>: Simulations-Server

Der Simulations-Server hat von einer äußeren Sicht zwei Teile. Der primäre Teil ist für alles rund um die Simulation und Datensammlung zuständig. Der sekundäre Teil ist eine REST-API, welche der Simulation aufliegt und dafür zuständig ist mit der Außenwelt zu kommunizieren. Beide Teile wurden hauptsächlich mit Python implementiert, wobei bei der Simulation noch ein kleiner Teil in Lua geschrieben wurde.

5.4.1 Teil 1: Simulation

Die Simulation benutzt drei externe Bibliotheken:

1. beamngpy (Schnittstelle für die Kommunikation mit dem BeamNG-Simulator)
2. keyboard (Zur Aufzeichnung von Aktionen auf der Tastatur)

3. pygltflib (Zum Umgang mit .gltf und/oder .glb Formaten)

Die Klasse *Simulator* ist der Hauptknotenpunkt. Wenn diese erstellt wird, läd sie erstmal alle Konfigurationen aus einer Datei. Das laden der Konfigurationen übernimmt dabei eine statische Klasse namens *ConfigHandler*.
Der *ConfigHandler* kann Konfigurationen laden und speichern, wobei letzteres praktisch nicht benutzt wird. Besonders an dem *ConfigHandler* ist, dass er in den Konfigurationen auch den Datentyp speichert und auch wieder auslesen kann. Das erleichtert den Umgang in Python.
Nachdem die Konfigurationen geladen sind, werden mehrere Objekte erstellt, besonders wichtig dabei ist die BeamNGpy-Instanz (des weiteren gibt es noch Objekte für das Logging). Die BeamNGpy-Instanz kommt aus der Bibliothek und ermöglicht die Kommunikation mit dem Simulator.
Zuletzt wird noch ein *CrashScenario* erstellt, welches ein Szenario in dem Simulator beschreibt, welches simuliert werden soll. Diese Klasse kümmert sich um den Aufbau der Szene, der relevanten Objekte (Auto und Sensoren) und auch die Ausführung der Simulation. Ein *CrashScenario* ist in sich abgeschlossen, d.h., es kann beliebig oft neu gestartet werden. Zum Starten der Simulation stellt es eine Methode nach außen bereit, welche dann auch die Ergebnisse der Simulation zurückliefert. Ebenfalls erwähnenswert, ist der Teil in dem das *CrashScenario* das 3D-Modell des Unfalls exportieren möchte. Da BeamNGpy dafür keinen Weg bereitstellt, musste ein extra Mod in Lua programmiert werden, welcher direkt in dem BeamNG-Programm läuft und damit mehr Zugriffsmöglichkeiten hat. Das *CrashScenario* stellt also eine Anfrage zum exportieren des 3D-Modells.
Der Lua-Mod (genannt *meshexport*) fängt diese Anfrage ab und nutzt direkt einen weiteren Lua-Mod (*util_ export* kommt mit dem BeamNG-Programm), welcher das 3D-Modell asynchron generiert. Wenn die Generierung dann abgeschlossen ist gibt *meshexport* das 3D-Modell zurück, welches dann von *CrashScenario* angenommen wird.

5.4.2 Teil 2: REST-API

Die REST-API ist grundsätzlich sehr überschaubar, denn sie akzeptiert nur eine Anfrage und das ist eine Get-Request, welche vier Paramter braucht (force, speed, hsn, tsn). Aus der HSN/TSN wird ein Autotyp ausgelesen. Dazu haben wir für jeden Auto-Typ in der Simulation einen Beispiel-Modell rausgesucht, welches dann erkannt wird. Später müsste man diesen Teil der Implementierung noch an eine Datenbank anschließen, welche für eine gegebene HSN/TSN den Auto-Typ auslesen kann. Der Auto-Typ und die Geschwindigkeit (speed) werden der Simulation übergeben. Daraufhin wird eine Simulation gestartet, welches eine Mesh zurückliefert. Das Mesh wird dann als Response für die Get-Request zurückgegeben.

5.4.3 Paket-/Klassendiagramm

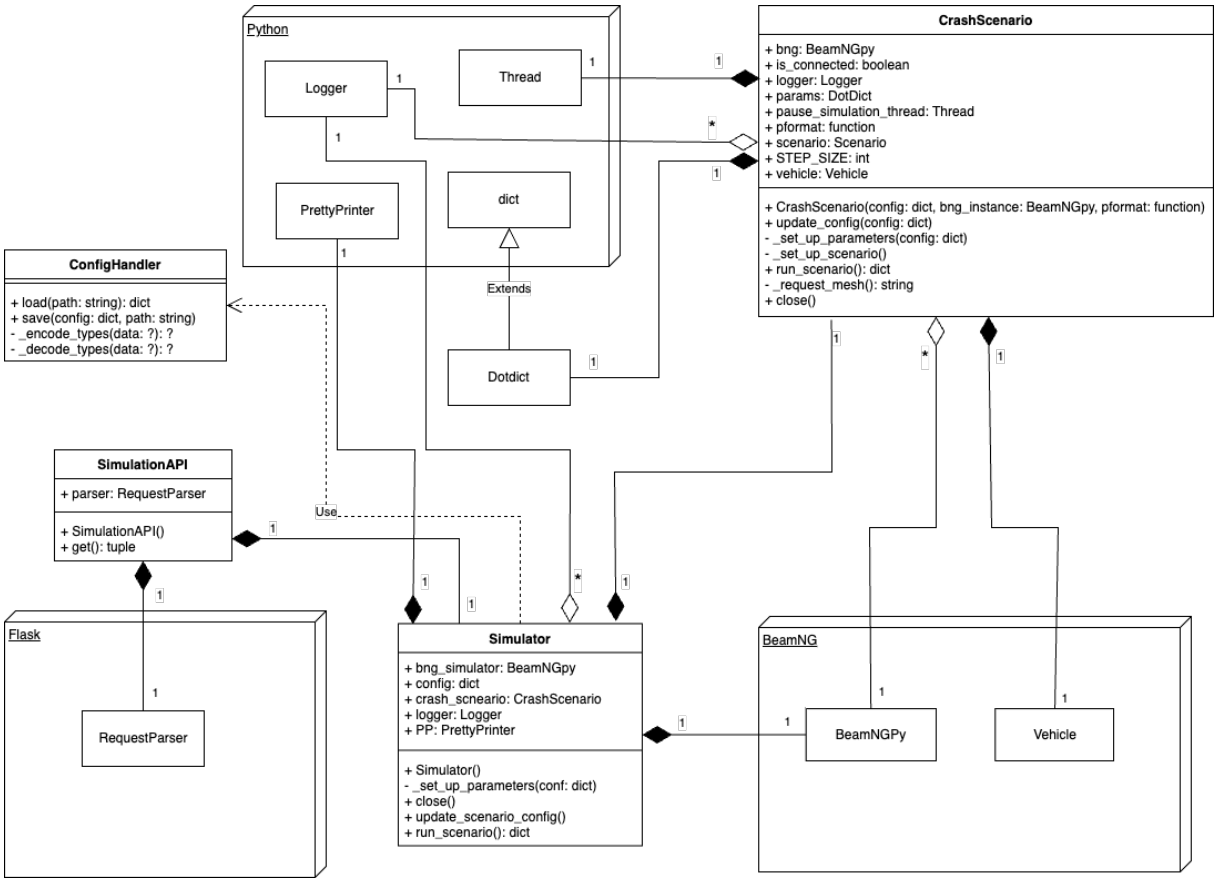

Abbildung 5.4: Klassendiagramm für Komponente $\langle C40 \rangle$ (erstellt mit draw.io)

5.4.4 Erläuterung

Zur Erläuterung des Klassendiagramms lässt sich die *Dokumentation der Implementierung* lesen.

6 Datenmodell

Im folgenden Kapitel wird das Datenmodell mithilfe eines UML-Klassendiagramms dargestellt. Die Anwendung wird die Unfalldaten lokal in einem .json Datei gespeichert. Neue Unfalldaten werden dann in dieser Datei mit Hilfe von REST-APIs hinzugefügt. Die Daten werden in zwei Entitäten Accident und Car aufgeteilt. Die Unfall Entität speichert Informationen über den Unfall wie die Zeit, Datum, Liste von beteiligten Autos und kann dann mit der zugehörigen ID des Unfalls identifiziert werden. Die Car-Entität erhält Informationen über die beteiligten Autos des Unfalls z.B. die geografische Position des Autos und die relevanten Informationen von dem Auto wie die Schlüsselnummer bzw. HSN/TSN. Diese werden auch in Car gespeichert, damit die Autos auf die Karte angezeigt werden können und die Informationen auf der Webseite zur Verfügung gestellt werden können. Für die Simulation wird zusätzlich noch die Geschwindigkeit, die Kraft und die Anzahl der Passagiere gebraucht. Ein Unfall bzw. Accident muss mindestens ein Auto bzw. eine Car Entität enthalten, wobei ein Auto nur einem Unfall zugeordnet werden kann

6.1 Diagramm

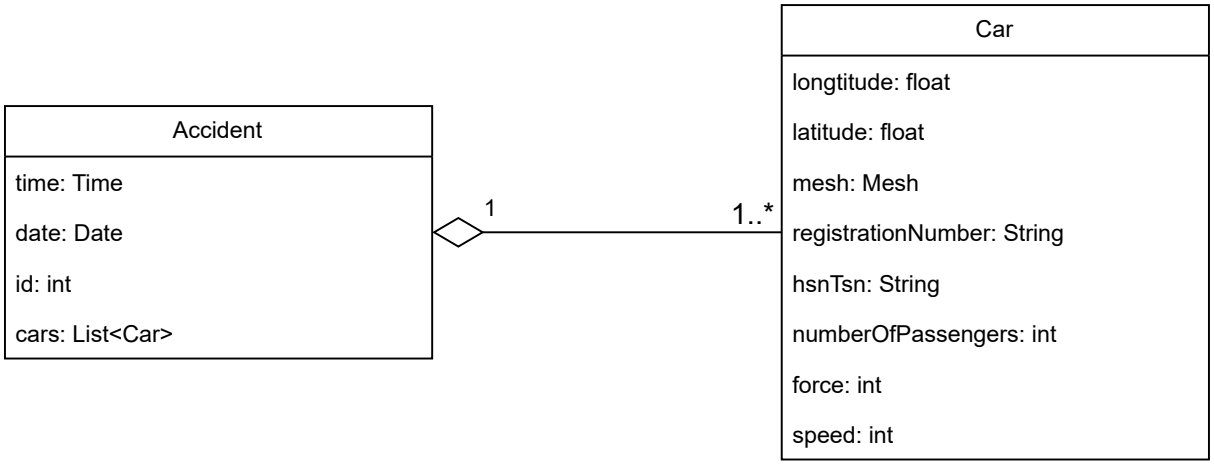

Abbildung 6.1: Klassendiagramm Produktdaten

6.2 Erläuterung

In der folgenden Tabelle wird die Beziehung zwischen Accident und Car erläutert.

Accident $\langle E10 \rangle$

Beziehung	**Kardinalität**	**Erwartete Datenmenge**	**Beschreibung**
Car	1, n	Min: 1, Max: 30, Pro Car etwa. 18 MB	Ein Accident kann beliebig viele Cars enthalten aber mindestens ein Car.

Car $\langle E20 \rangle$

Beziehung	**Kardinalität**	**Erwartete Datenmenge**	**Beschreibung**
Accident	1	Pro Car etwa. 18 MB	Ein Auto kann nur in einem Unfall enthalten.

7 Konfiguration

Dieses Kapitel enthält alle erforderlichen Konfigurationen für die Simulationsumgebung, die auf der BeamNG-Anwendung basiert, sowie für die Entwicklungsumgebung der Website, in der Typescript und das Angular-Frontend-Framework verwendet werden. Darüber hinaus wird die Konfiguration der Hardware, insbesondere des PCs, beschrieben, da dies für einen reibungslosen Betrieb der Anwendung erforderlich ist.

7.1 Simulationsumgebung

Für die Simulation wird die Plattform BeamNG in der Version BeamNG.tech v0.28.1.0 verwendet. Eine präzise Konfiguration für die Simulation und das Logging ist in der Datei „basicConfig.json“ enthalten. Die Konfiguration umfasst wie die Startposition und -rotation des Fahrzeugs sowie weitere Konstanten wie die Geschwindigkeit beim Aufprall. Bestimmte Parameter für das Logging sind ebenfalls in der Konfigurationsdatei definiert.

7.2 Webanwendung

Die Webanwendung setzt auf zwei Schnittstellen, nämlich den Server und die Website. Die Kommunikation zwischen diesen erfolgt mittels REST APIs. Zur Entwicklung der Website werden Typescript und das Angular-Framework benötigt. Das REST-API baut auf Node.js und dem Express-Framework auf. Die Konfigurationsdatei "config.js"beinhaltet die URL, welche für die Simulation verwendet wird.
Typescript ist eine freie und Open-Source-Programmiersprache, die von Microsoft entwickelt wurde. Sie erweitert JavaScript um zusätzliche Funktionen wie Typisierung und Klassendefinitionen. Für die vorliegende Anwendung ist die Version 4.9.3 erforderlich. Zur Konfiguration sind folgende Schritte in der Stammdatei vorzunehmen:

- Die Datei „tsconfig.json“ dient dazu, grundlegende TypeScript- und Angular-Compileroptionen festzulegen, welche von allen Projekten im Arbeitsbereich geerbt werden.
- Die Datei „tsconfig.app.json“ ermöglicht eine Anpassung der Konfiguration auf App-Basis.

- Die Datei „tsconfig.spec.json“ stellt eine spezifische TypeScript-Konfiguration für die Anwendungstests bereit.

Angular ist eine in TypeScript eingebettete Webentwicklungsplattform, die Entwicklern robuste Werkzeuge für die Erstellung der Clientseite von Webanwendungen zur Verfügung stellt. Für die vorliegende Anwendung ist die Version 16.0.0 erforderlich. Die Konfigurationsdatei hierfür ist die „angular.json“ im JSON-Format. Diese Datei speichert Informationen über die Architektur des Projekts, Abhängigkeiten, Build- und Testkonfigurationen sowie weitere Einstellungen.

7.3 PC/Server

Für eine reibungslose und effiziente Entwicklung der Simulation und Website sind spezifische Hardwareanforderungen zu berücksichtigen, die u.a. die Prozessorgeschwindigkeit, den Arbeitsspeicher, die Grafikkarte und die Festplattenkapazität umfassen. Aus diesem Grund ist ein leistungsstarker PC erforderlich, der mit Windows 10 als Betriebssystem ausgestattet ist. Der PC dient als zentraler Bestandteil der Applikation, indem er als Server fungiert. Desweiteren fungiert er als Datenspeicher.
Für die Ausführung der Webanwendung wurde NidaPad als geeignete Plattform ausgewählt. Die Entscheidung für NidaPad gründet sich auf der robusten, skalierbaren und mobilen Infrastruktur, die es für die Kommunikation in Notfällen bereithält.

8 Änderungen gegenüber Fachentwurf

In diesem Kapitel werden die Änderungen aufgelistet, die gegenüber dem Fachnetwurf aufgetreten sind.

8.1 Analyse der Produktfunktionen

Die Funktion <F9> wurde von uns nicht entwickelt, da am Ende die Zeit leider nicht reichte und uns, nachdem wir uns näher mit dieser Funktion befasst haben, klar wurde, dass der entwicklungstechnische Aufwand ziemlich groß ist. Dennoch erläutern wir im folgenden, wie diese Funktion implementiert werden könnte: Um in der Simulation die Krafteinwirkung auf mögliche Insassen zu berücksichtigen gibt es grundsätzlich zwei Möglichkeiten. Die erste besteht daraus, dass man Dummys in das Fahrzeug setzt und dann während der Simulation die Krafteinwirkung auf diese Dummys ausliest. Der Aufwand der Entwicklung ist hier sehr groß, denn BeamNG unterstützt von Haus aus keine Ragdoll-Physik, d.h. man müsste eine Art Mod entwickeln, welcher dies ermöglicht. Die zweite Möglichkeit wäre mit IMU-Sensoren am Fahrzeug. Diese IMU-Sensoren lassen sich in beliebiger Anzahl an festen Position am Fahrzeug positionieren. Während der Simulation kann man dann zu jedem Zeitpunkt die Krafteinwirkung auf diesen Punkt auslesen. Man könnte nun für jeden Sitz im Auto mehrere IMU's an den jeweiligen Körperstellen positionieren und daraus am Ende und damit am Ende eine Krafteinwirkung auf eine gesamte Person betrachten. Der Umfang ist hier auch wieder relativ hoch, da man sehr viele Positionen im Auto finden und definieren muss. Der Aufwand wird noch dadurch erhöht, dass wir mehrere Auto-Modelle für die Simulation haben, wodurch sich natürlich auch die Positionen der IMU's unterscheiden würden.

Hinweis: Auf Grund von Einrichtungsschwierigkeiten des Google-Kontos, welches für die Erstellung der 3D-Visualisierung mittels der Google Maps JavaScript API zwingend benötigt wird, konnten die Funktionen, die die Karte betreffen noch nicht programmiert werden. Wie besprochen wird nur der aktuell existierende Programmcode in den Kapiteln dargestellt.

9 Erfüllung der Kriterien

Im Folgenden werden kurz die Muss-, Soll- und Kannkriterien aus dem Pflichtenheft betrachtet und die Komponenten dargestellt, durch die sie umgesetzt werden.

9.1 Musskriterien

Die folgenden Kriterien sind unabdingbar und müssen durch das Produkt erfüllt werden:

RM1 *Die Anwendung muss ISANs entgegennehmen können und die Daten auswerten können.* Dieses Kriterium wird von der Komponente <C30> Datenauswertung bearbeitet.
RM2 *Die ISANs werden gespeichert. Zeitlich und geografisch nahe beieinanderliegende Nachrichten werden zu einem Unfall mit eindeutiger ID zusammengefasst.* Dies wird durch die Komponente <C40> Datenverarbeitung umgesetzt.
RM3 *Der Schaden des Fahrzeuges wird mittels BeamNG und den Informationen aus der ISAN auf einem Server simuliert.* Dieses Kriterium wird von der Komponente <C10> Simulation und <C30> Datenauswertung übernommen.
RM4 *Die Simulation des Unfalls liefert 3D-Modelle der beschädigten Fahrzeuge zurück.* Dies übernimmt ebenfalls die Simulation <C10>.
RM5 *Es gibt eine Website mit eingebetteter Karte, die die Modelle entgegennimmt und an den Koordinaten des Unfalls anzeigt.* Dieses Kriterium wird von der Komponente <C20> GUI übernommen.
RM6 *Es gibt eine Startseite mit einem Suchfeld, in welches die ID des Unfalls eingegeben werden kann und man dann die Kartenansicht mit den Fahrzeugen des Unfalls und weiteren Informationen erhält.* Dieses Kriterium wird von den Komponenten <C20> GUI und <C30> Datenauswertung umgesetzt.
RM7 *Ein Unfall wird vom Server nach einer fest definierten Zeit gelöscht. Damit werden alle dazugehörigen Daten und Modelle von der Karte entfernt und auf dem Server gelöscht.* Dieses Muss-Kriterium wird von der Komponente <C40> Datenverarbeitung umgesetzt.
RM8 *Es wird zum ausgewählten Fahrzeug ein Link zu der entsprechenden Rettungskarte, die sich in einem neuen Fenster öffnet, bereitgestellt.* Dieses Kriterium wird von der Komponente <C20> GUI übernommen.

9.2 Sollkriterien

Die Erfüllung folgender Kriterien für das abzugebende Produkt wird angestrebt:

RS1 *Das Erscheinungsbild der Website soll ansprechend, übersichtlich, anwenderfreundlich und intuitiv gestaltet sein.* Dieses Kriterium wird von der Komponente <C20> GUI übernommen.
RS2 *Wenn der Mauszeiger sich über einem Fahrzeug befindet, soll ein Fenster eingeblendet werden mit Informationen zu Krafteinwirkung und Insassenanzahl.* Dies übernimmt ebenfalls die GUI <C20>.

9.3 Kannkriterien

Die Erfüllung folgender Kriterien für das abzugebende Produkt wird angestrebt:

RC1 *Bei der Auswahl eines Fahrzeuges werden die Krafteinwirkungen als Heatmap/Vertex auf die Insassen angezeigt.* Dieses Kriterium wird von den Komponenten <C20> GUI und <C10> Simulation umgesetzt.
RC2 *Die Anzeige der Krafteinwirkungen auf die Insassen erfolgt entweder abhängig vom ausgewählten Fahrzeug neben der Karte oder wird als extra Fenster eingeblendet.* Dieses Kriterium wird von der Komponente <C20> GUI übernommen.
RC3 *Die Darstellungen der Insassen mit Krafteinwirkung sollen sich drehen lassen, sodass sich auch die Krafteinwirkung auf der Rückseite der Figuren erkennen lässt.* Dieses Kriterium wird von der Komponente Simulation <C10> übernommen.
RC4 *Die Verlinkung zur Rettungskarte kann entweder für jedes Fahrzeug des Unfalls fest eingeblendet werden oder es wird nur die Verlinkung passend zum aktuell ausgewählten Fahrzeug angezeigt oder die Verlinkung ist in dem Fenster mit Kraft und Insassenanzahl enthalten, welches sich öffnet, wenn der Mauszeiger über dem Fahrzeug schwebt.* Dieses Kriterium wird von der Komponente GUI <C20> bearbeitet.

10 Glossar

Angular ist ein TypeScript-basiertes Front-End-Webapplikationsframework. Es wird von einer Community aus Einzelpersonen und Unternehmen, angeführt durch Google, entwickelt und als Open-Source-Software publiziert.

BeamNGpy ist eine offizielle Bibliothek, die eine Python-API für BeamNG.tech bereitstellt, den akademie- und industrieorientierten Fork des Videospiels BeamNG.drive.

Frontend und **Backend** werden in der Informationstechnik an verschiedenen Stellen in Verbindung mit einer Schichteneinteilung verwendet. Dabei ist typischerweise das Frontend näher am Benutzer, das Backend näher am System.

glTF ist die Kurzform von Graphic Language Transmission Format. Es beinhaltet dreidimensionale Szenen und Modelle. Eine glTFDatei beinhaltet entweder ein glTF (JSON/ ASCII) oder ein GLB (binär) als mögliche Dateierweiterung.
HSN und TSN ergeben zusammen einen alphanumerischen Code, der nicht nur Zahlen, sondern auch Buchstaben enthält. Die Kombination aus HSN und TSN gibt unter anderem Auskunft über den Fahrzeugtyp, die Motorleistung, den Hubraum und die Art des Kraftstoffes. **HTML** ist eine textbasierte Auszeichnungssprache zur Strukturierung elektronischer Dokumente wie Texte mit Hyperlinks, Bildern und anderen Inhalten.

ISAN (International Standard Accident Number) Das ISAN-Projekt zielt darauf ab große Datenmengen aus diversen Quellen, wie der Notfallmedizin (EMS), der elektronischen Gesundheitsakte (EHR) und Ereignisdatenschreibern (EDR) zu sammeln und durch die Schaffung einer technischen Grundlage zu verbinden und Rettungseinsätze zu unterstützen. Die ISAN selbst enthält Unfalldaten.

Klasse in der objektorientierten Programmierung ist ein abstraktes Modell bzw. einen Bauplan für eine Reihe von ähnlichen Objekten.

NidaPad ist ein Tabletcomputer, der gezielt für Rettungskräfte und Einsatzbedingungen entwickelt wurde. Das NIDApad unterstützt die Kommunikation zwischen Leitstelle und Rettungswagen, die Navigation zum Einsatzort, eine umfassende Einsatzdokumentation sowie die Wei-

tergabe der Einsatzdaten an die Klinik.

Node.js ist eine Single-Thread, Open Source, plattformübergreifende Laufzeitumgebung für die Erstellung von schnellen und skalierbaren serverseitigen und Netzwerkanwendungen.

Python ist eine einfach zu erlernende, interpretierte, objektorientierte Programmiersprache mit dynamischer Typisierung.

Rettungskarte ist eine bereits vorhandene Softwarekomponente, die Informationen eines spezifischen Fahrzeugs enthält, welche den Rettungskräften helfen, das Fahrzeug gefahrlos zu öffnen.

TypeScript ist eine kostenlose und Open-Source-Programmiersprache, die auf JavaScript aufbaut und statische Typisierung unterstützt.

3D-Mesh ist eine dreidimensionale Struktur, die eine 3D-Objekt darstellt.