Webanwendung zur Visualisierung von rettungsrelevanten Daten zur Einsatzunterstützung

Software-Entwicklungspraktikum (SEP)
Sommersemester 2023

Fachentwurf

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, Mohamed Wassim Chebili	keine
1.1	Omar Farouk Khayat, Mohamed Wassim Chebili	keine
1.2	Omar Farouk Khayat, Mohamed Wassim Chebili	keine
2	Jonas Stepanik	keine
2.1	Jonas Stepanik	keine
2.2	Torben Oelerking	keine
2.3	Jonas Stepanik	keine
2.4	Torben Oelerking	keine
2.5	Torben Oelerking	keine
2.6	Kacem Abdennabih	keine
2.7	Kacem Abdennabih	keine
2.8	Kacem Abdennabih	keine
2.9	Jonas Stepanik	keine
3	Azhar Rahadian	keine
3.1	Azhar Rahadian	keine
3.2	Azhar Rahadian	keine
4	Qiyue Zhang	keine
3.2	Qiyue Zhang	keine
3.2	Qiyue Zhang	keine
3.2	Qiyue Zhang	keine
5	zusammen	keine

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Dieses Dokument enthält eine detaillierte Beschreibung des Entwurfs der Funktionen des Produkts. Im ersten Kapitel wird eine umfassende allgemeine Beschreibung des Projekts gegeben, wobei insbesondere auf komplexe Sachverhalte eingegangen wird. Es werden verschiedene Aspekte und Hintergründe erläutert, um ein grundlegendes Verständnis für das Projekt zu vermitteln. Im zweiten Kapitel erfolgt eine eingehende Analyse der Produktfunktionen. Hier werden die Funktionen im Detail betrachtet und mithilfe von Sequenzdiagrammen visualisiert. Dadurch wird ein klarer Einblick in die Abläufe und Interaktionen der verschiedenen Funktionen des Produkts ermöglicht. Das dritte Kapitel widmet sich den Beziehungen zwischen den Daten. Es werden die verschiedenen Datenpunkte und ihre Verbindungen dargestellt und erläutert. Dadurch wird ein umfassendes Verständnis für die Datenstruktur und deren Zusammenhänge geschaffen. Im vierten Kapitel werden die Konfigurationsdateien erläutert und ihre spezifischen Funktionen erklärt. Hier wird aufgezeigt, wie diese Dateien verwendet werden, um das Produkt anzupassen und bestimmte Einstellungen zu konfigurieren.

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.

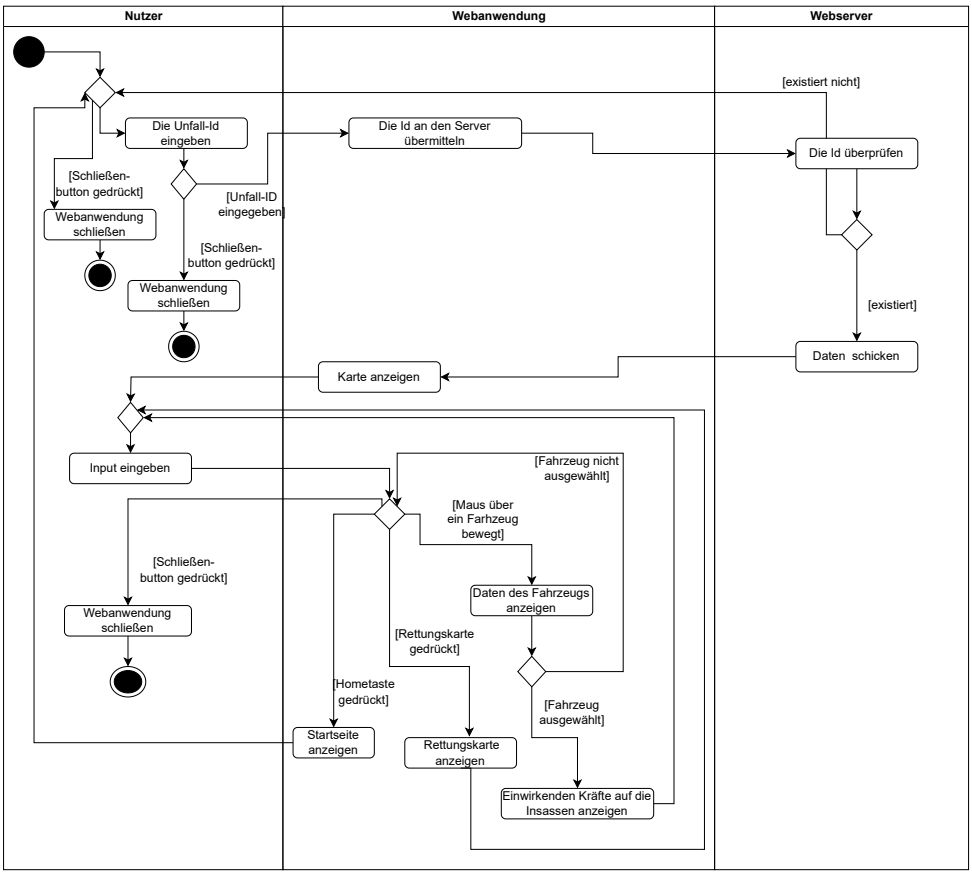

Abbildung 1.1: Aktivitätsdiagramm aus der Nutzersicht

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.

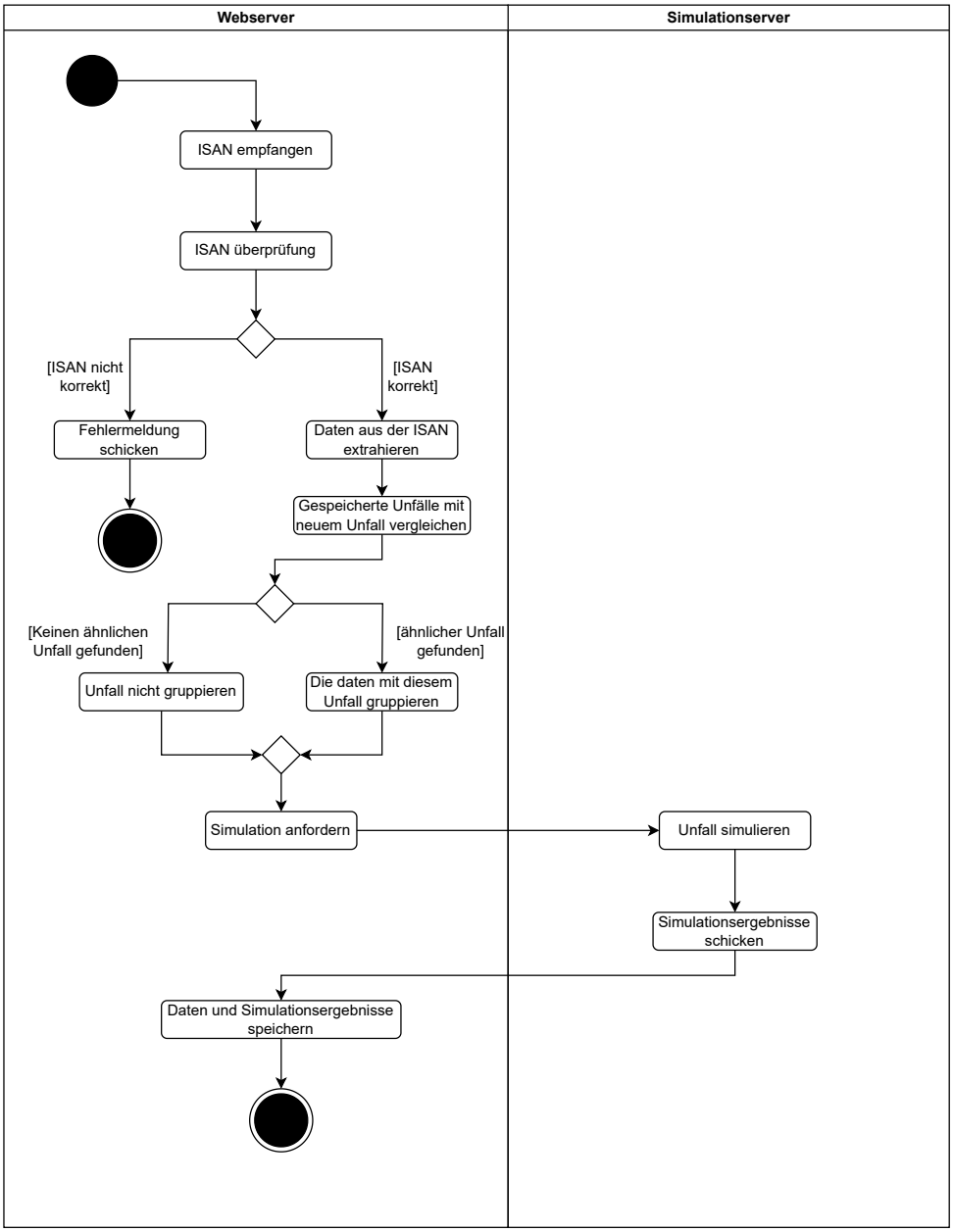

Abbildung 1.2: Aktivitätsdiagramm beim hinzufügen einen neuen Unfall

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.

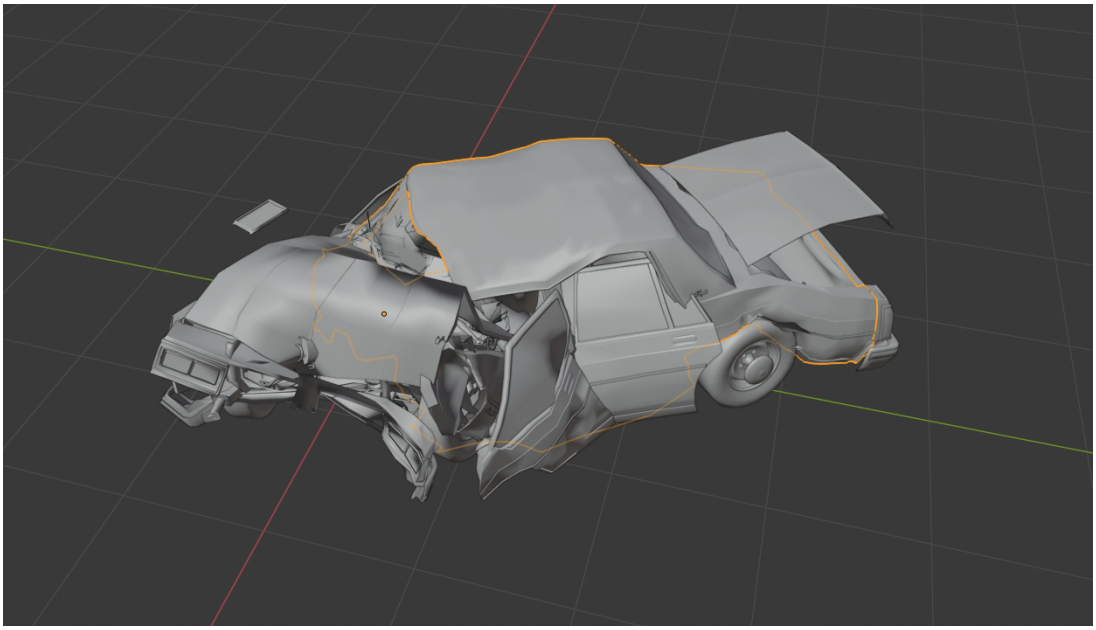

Abbildung 1.3: Ein simulierter Unfall

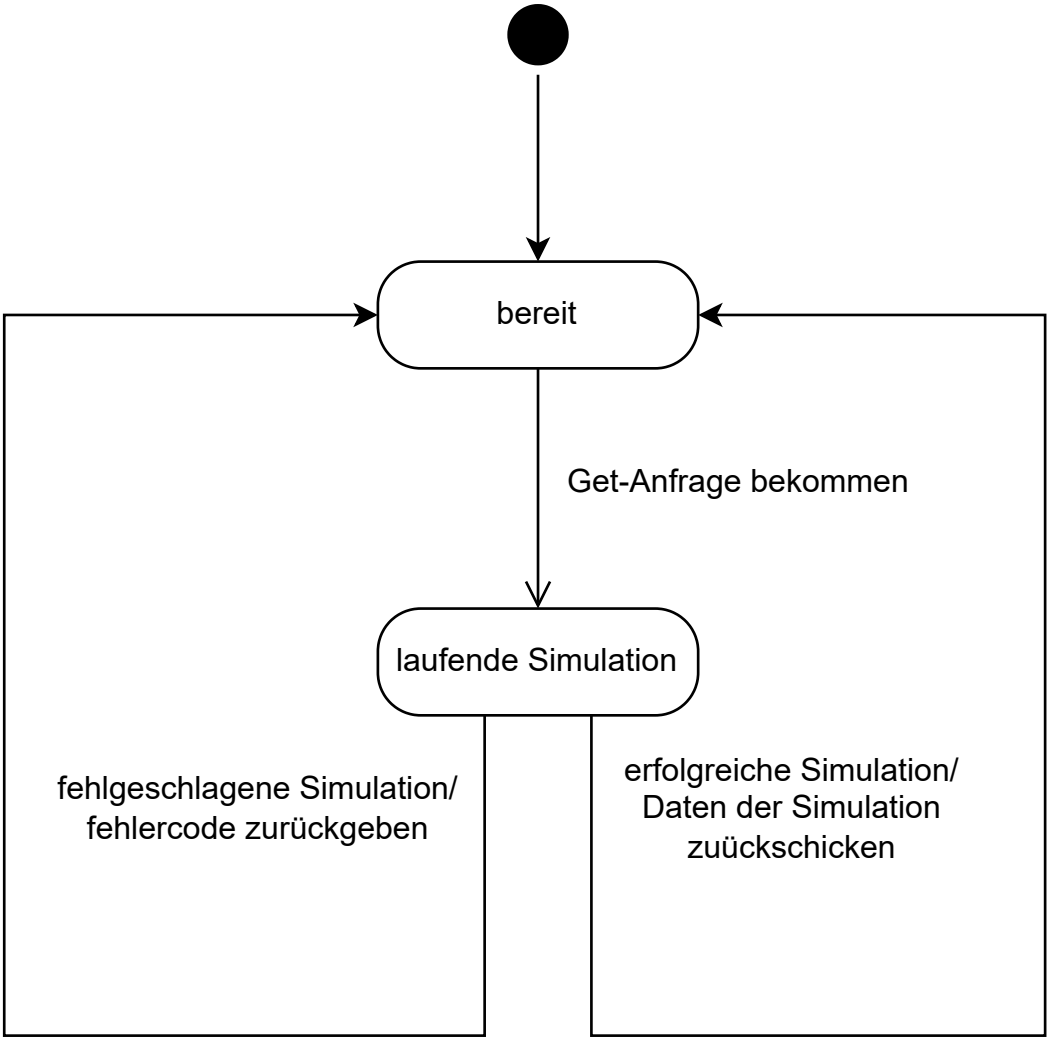

Abbildung 1.4: Zustanddiagramm der Simulation

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
    }
  ]
}
```

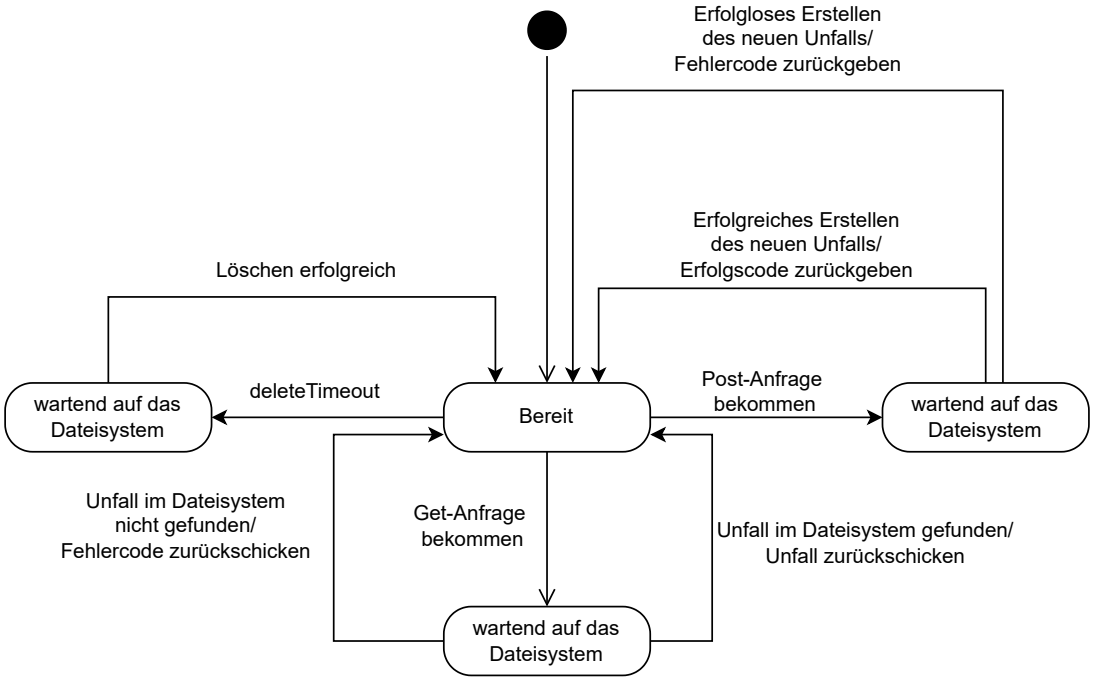

Abbildung 1.5: Zustanddiagramm des Webservers

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 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 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.

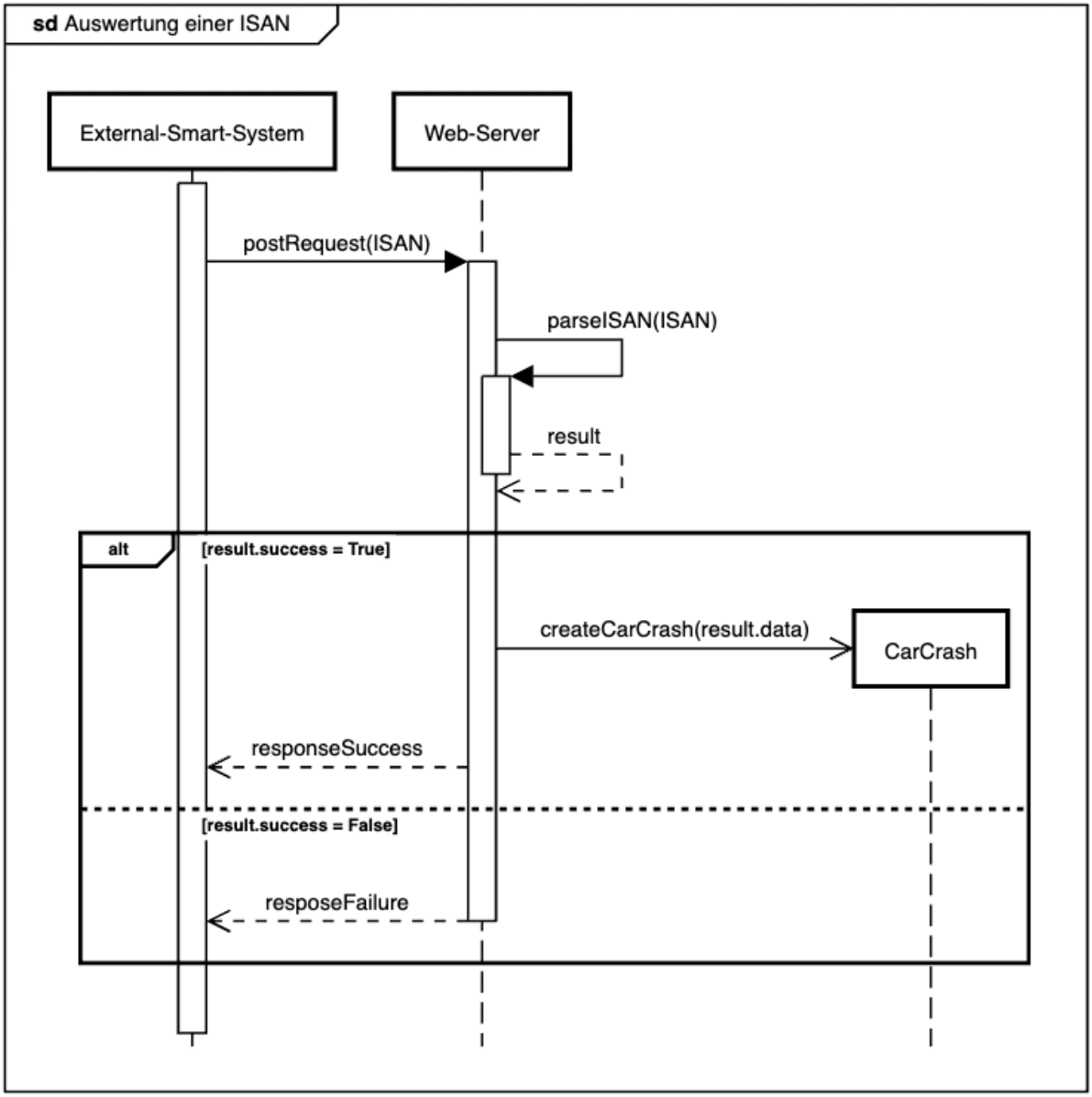

Abbildung 2.1: Auswertung einer ISAN (https://www.sequencediagram.org)

2.2 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.

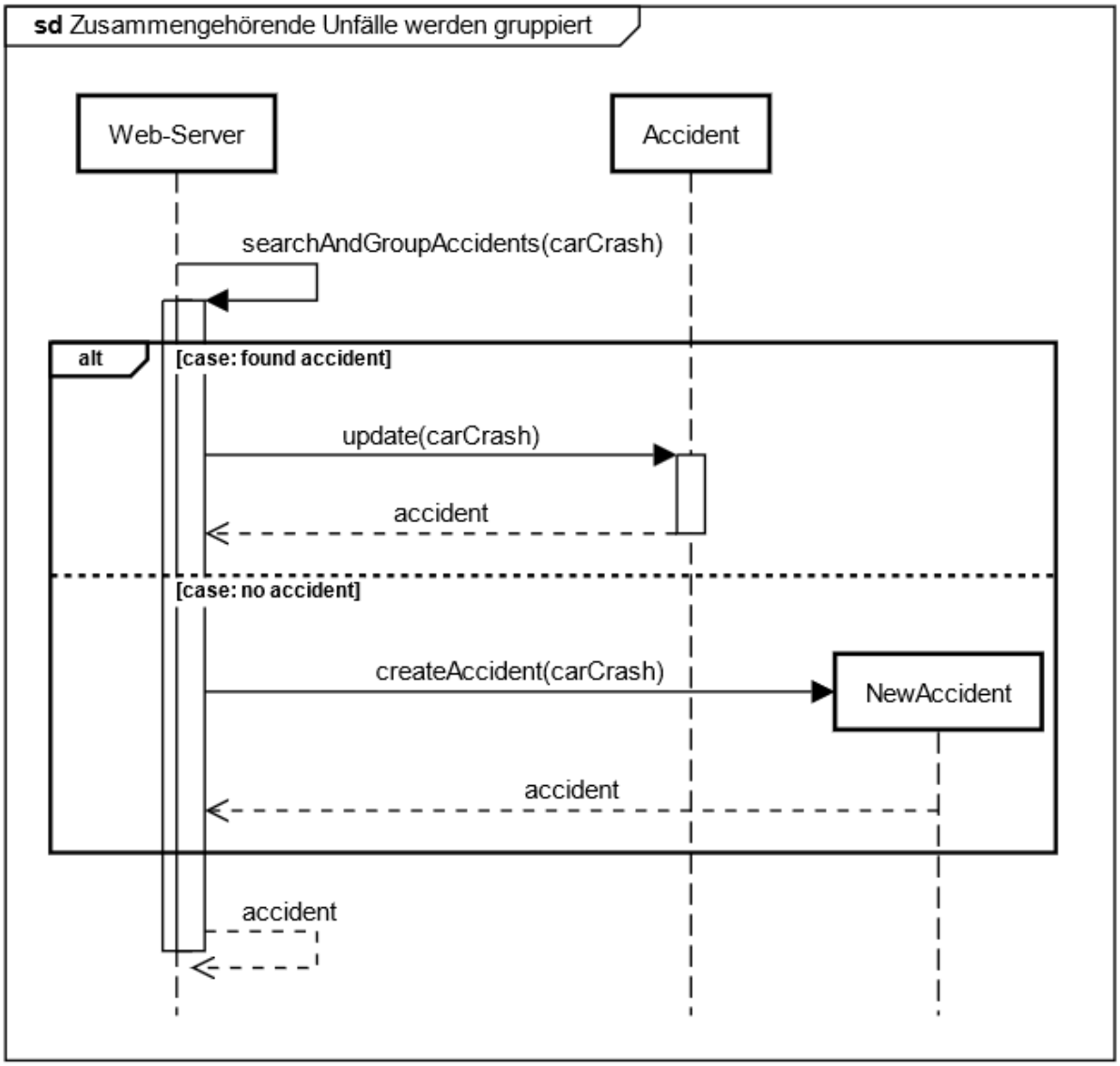

Abbildung 2.2: Zusammengehörende Unfälle werden gruppiert (https://www.sequencediagram.org)

2.3 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.

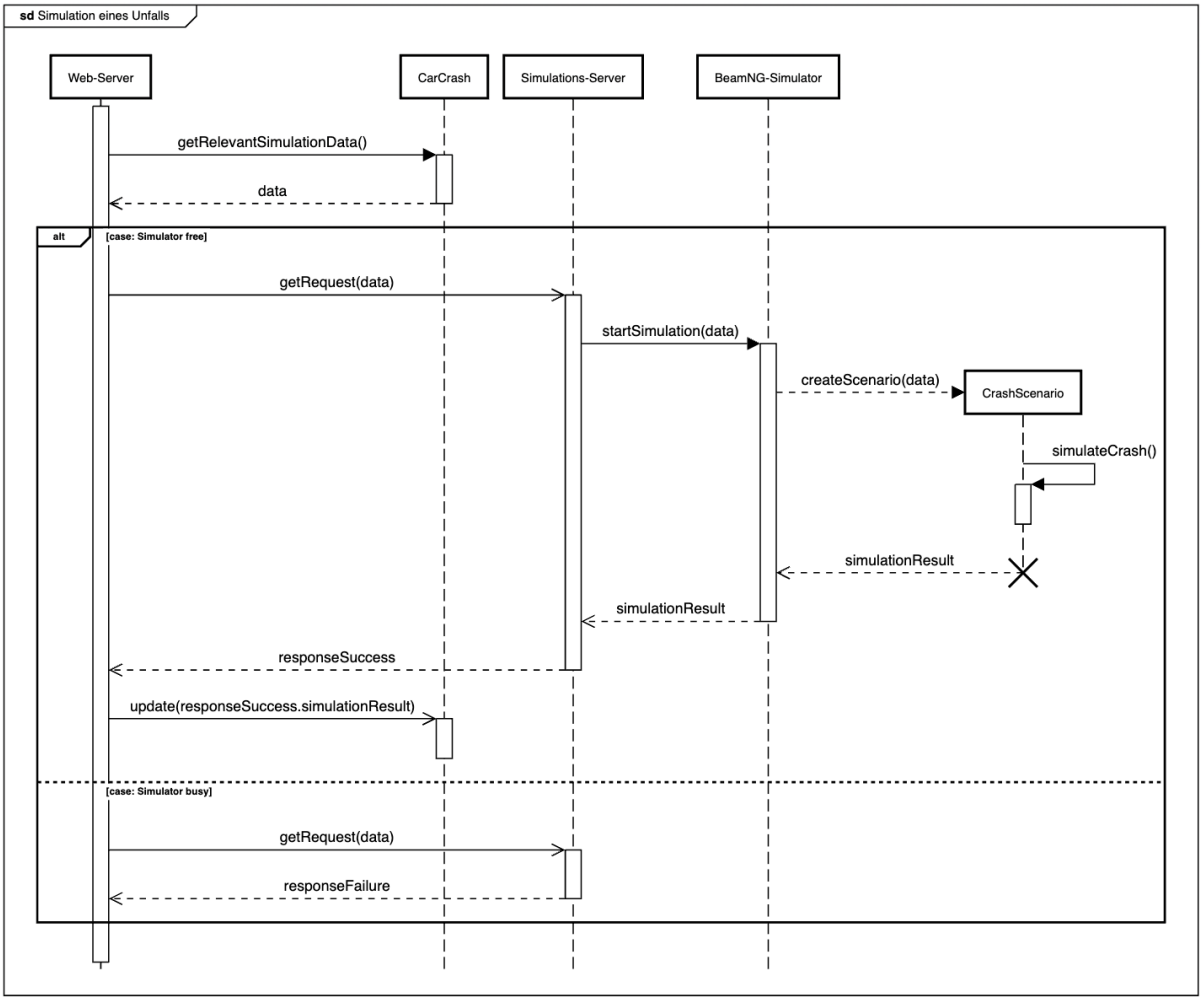

Abbildung 2.3: Simulation eines Unfalls (https://www.sequencediagram.org)

2.4 Analyse von Funktionalität <F4>: Visulisierung 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.

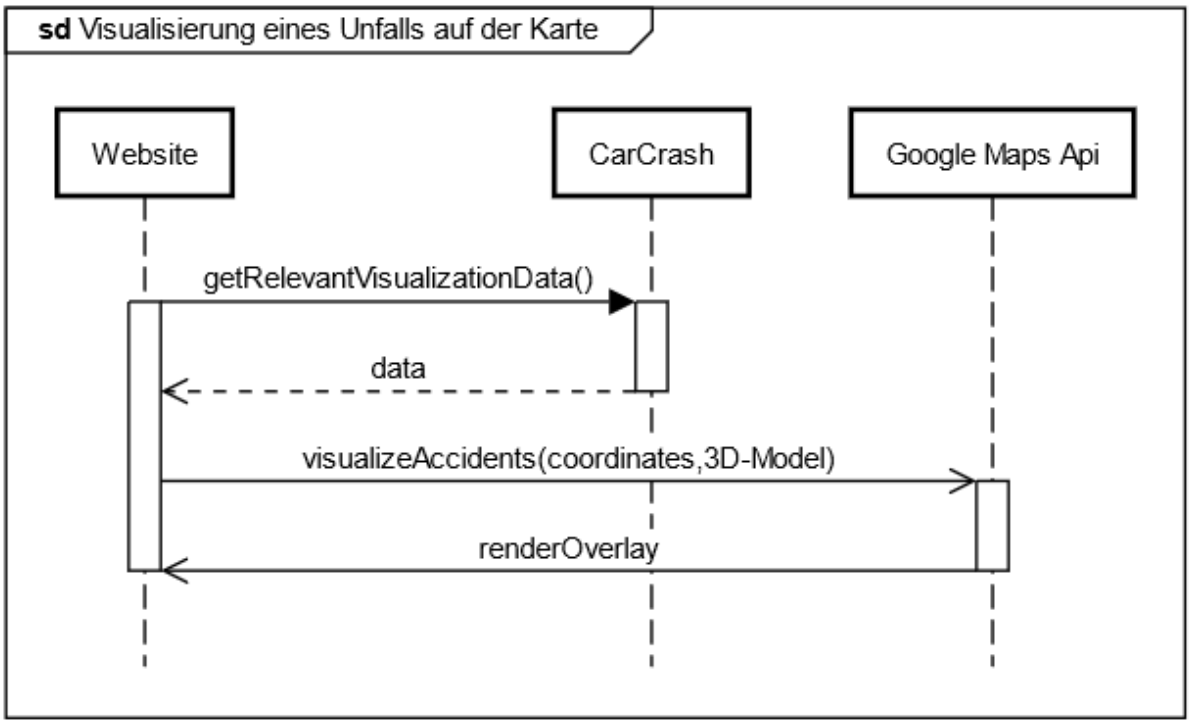

Abbildung 2.4: Visualisierung eines Unfalls auf der Karte (https://www.sequencediagram.org)

2.5 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.

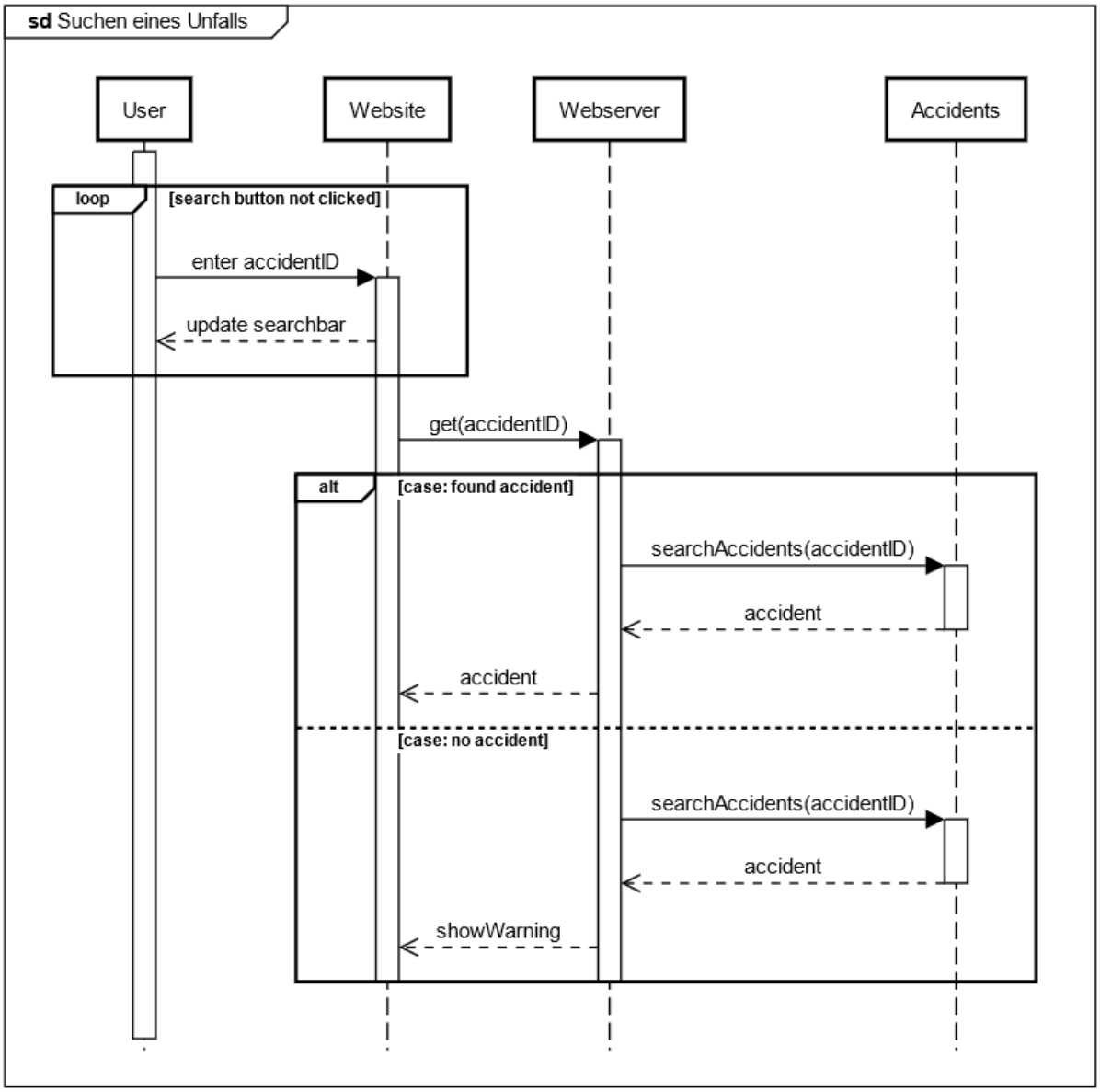

Abbildung 2.5: Suchen eines Unfalls (https://www.sequencediagram.org)

2.6 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()).

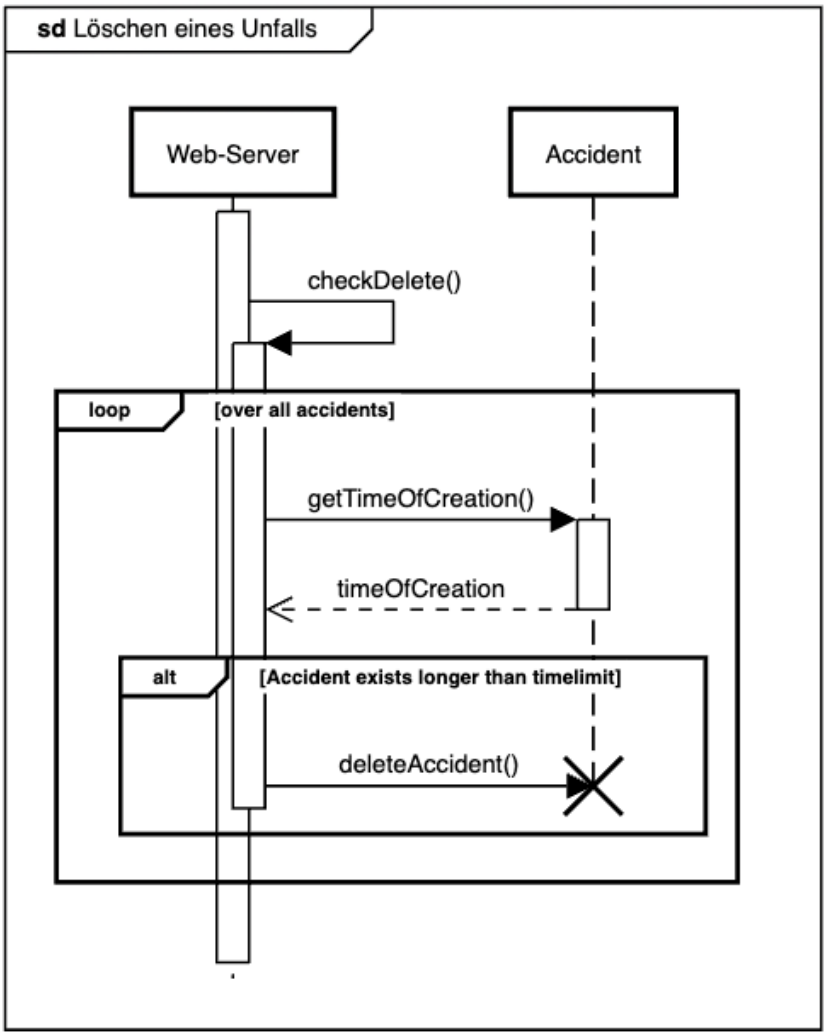

Abbildung 2.6: Löschen eines Unfalls (https://www.sequencediagram.org)

2.7 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.

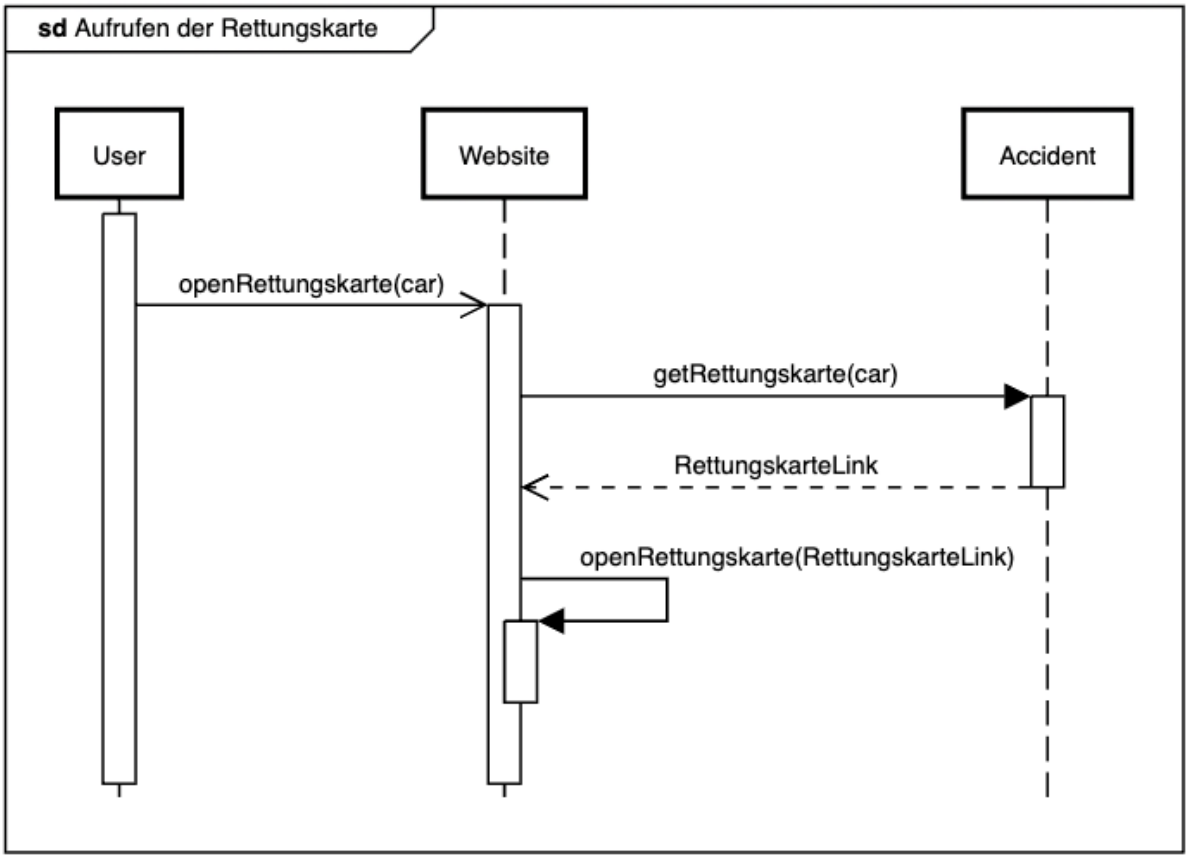

Abbildung 2.7: Aufrufen der Rettungskarte (https://www.sequencediagram.org)

2.8 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)).

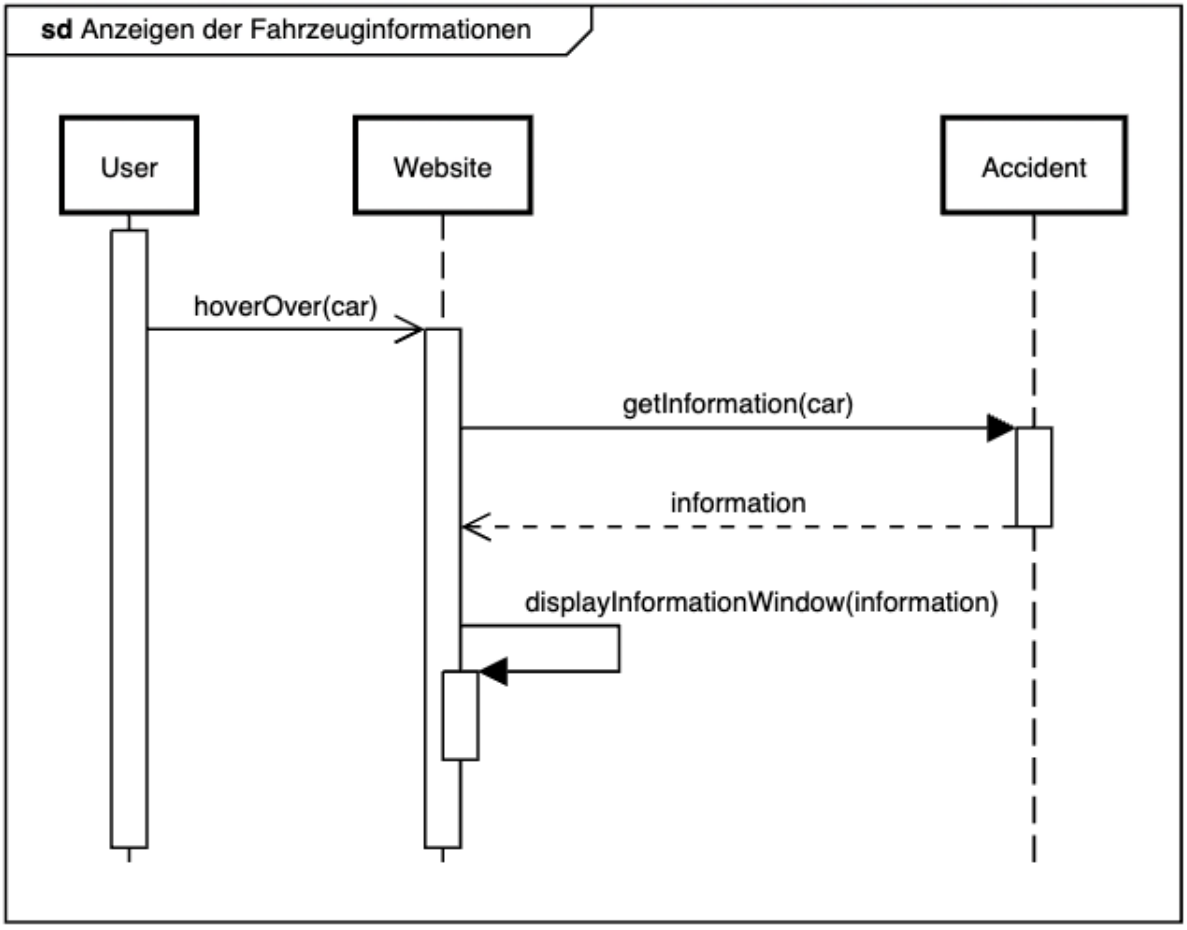

Abbildung 2.8: Anzeige der Fahrzeuginformationen (https://www.sequencediagram.org)

2.9 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.

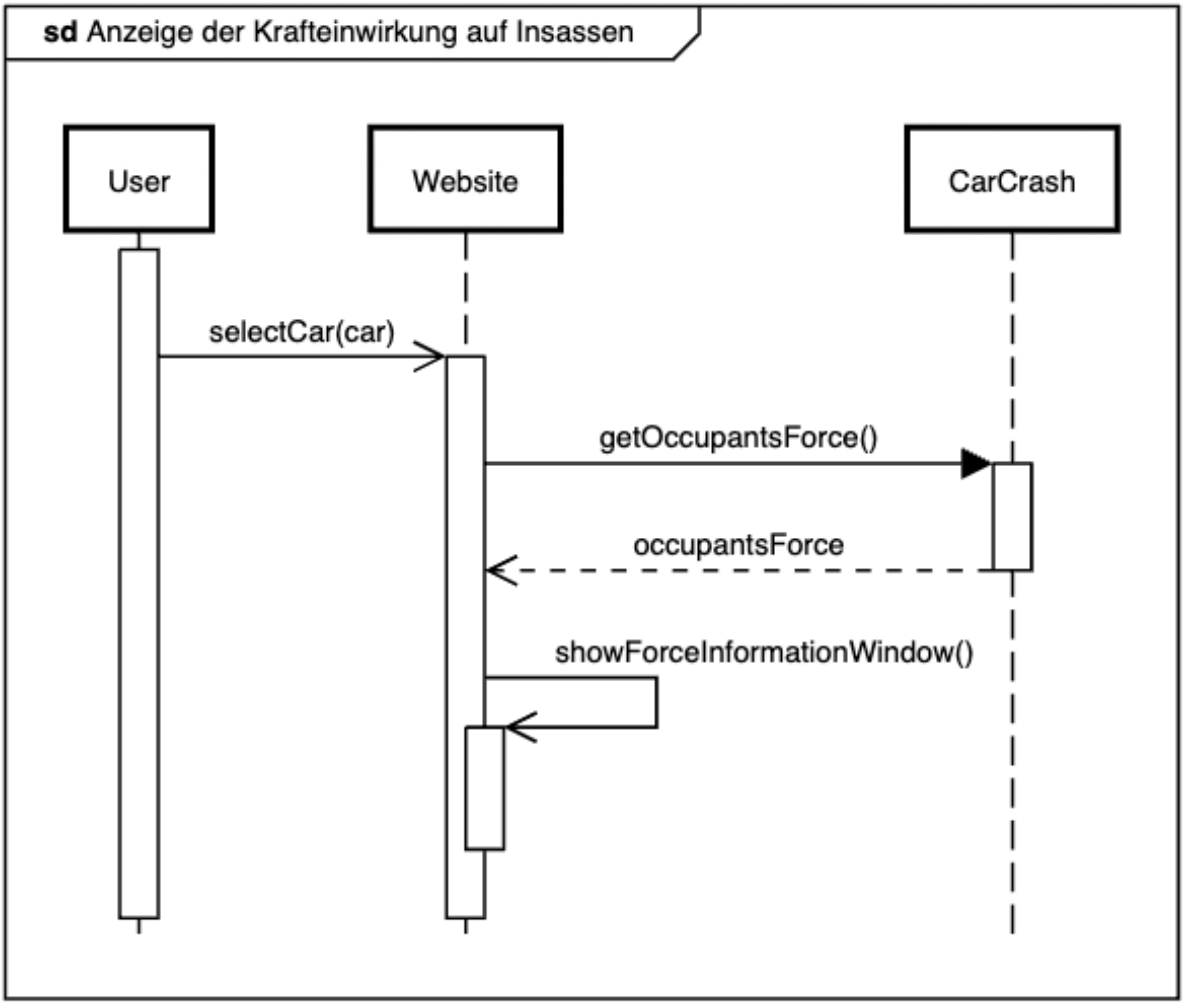

Abbildung 2.9: Anzeige der Krafteinwirkung auf Insassen (https://www.sequencediagram.org)

3 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.

3.1 Diagramm

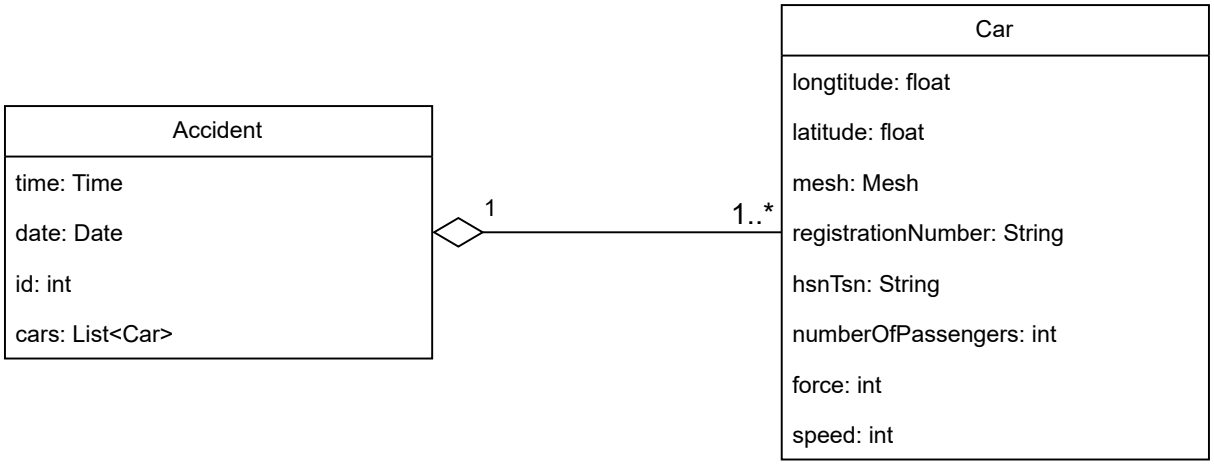

Abbildung 3.1: Klassendiagramm Produktdaten

3.2 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.

4 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.

4.1 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.

4.2 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.
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.

4.3 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.

5 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 Rettungs-

wagen, die Navigation zum Einsatzort, eine umfassende Einsatzdokumentation sowie die Weitergabe der Einsatzdaten an die Klinik.

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.