Webanwendung zur Visualisierung von rettungsrelevanten Daten zur Einsatzunterstützung

Software-Entwicklungspraktikum (SEP)
Sommersemester 2023

Testspezifikation

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, 12. Juli 2023

Bearbeiterübersicht

Kapitel	Autoren	Kommentare
1	Torben Oelerking	keine
2	Mohamed Wassim Chebili	keine
2.1	Mohamed Wassim Chebili	keine
2.2	Mohamed Wassim Chebili	keine
2.3	Mohamed Wassim Chebili	keine
2.4	Azhar Rahadian	keine
2.4.1	Azhar Rahadian	keine
2.4.2	Azhar Rahadian	keine
2.4.3	Azhar Rahadian	keine
2.4.4	Azhar Rahadian	keine
2.5	Azhar Rahadian	keine
3	Jonas Stepanik	keine
3.1	Jonas Stepanik	keine
3.2	Jonas Stepanik	keine
3.2.1	Jonas Stepanik	keine
3.3	Jonas Stepanik	keine
3.3.1	Jonas Stepanik	keine
3.3.2	Jonas Stepanik	keine
3.3.3	Jonas Stepanik	keine
3.3.4	Jonas Stepanik	keine
3.3.5	Jonas Stepanik	keine
3.3.6	Jonas Stepanik	keine
3.3.7	Jonas Stepanik	keine
3.3.8	Jonas Stepanik	keine
3.3.9	Jonas Stepanik	keine
3.3.10	Jonas Stepanik	keine
4	Torben Oelerking	keine
4.1	Omar Farouk Khayat	keine
4.2	Zusammen	keine
4.2.1	Torben Oelerking	keine

4.3	Mohamed Wassim Chebili	keine
4.3.1	Jonas Stepanik	keine
4.3.2	Omar Farouk Khayat	keine
5	Azhar Rahadian	keine
5.1	Alle	keine
5.2	Alle	keine
5.3	Zusammen	keine
5.3.1	Jonas Stepanik	keine
5.3.2	Jonas Stepanik	keine
5.3.3	Mohamed Wassim Chebili	keine
5.3.4	Mohamed Wassim Chebili	keine
5.3.5	Mohamed Wassim Chebili	keine
5.3.6	Omar Farouk Khayat	keine
5.3.7	Omar Farouk Khayat	keine
5.3.8	Kacem Abdennabih	keine
5.3.9	Kacem Abdennabih	keine
5.3.10	Qiyue Zhang	keine
5.3.11	Qiyue Zhang	keine
5.3.12	Qiyue Zhang	keine
5.3.13	Torben Oelerking	keine
6	Omar Farouk Khayat	keine

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Das Testen von Software ist essenziell wichtig, da so die Qualität der zu entwickelnden Software gewährleistet werden kann.
Das Testen der Software erfolgt bereits im Entwicklungsprozess, um Probleme frühzeitig zu erkennen, Verzögerungen und Gefährdungen des Projektes zu vermeiden, die Fehlerquote gering zu halten und die im Pflichtenheft formulierten nichtfunktionalen Anforderungen und Qualitätsmerkmale an das Produkt best möglich zu erfüllen und umzusetzen.
Um die Qualität der Software zu gewährleisten und entsprechend zu dokumentieren, orientiert sich die Testdokumentation an dem IEEE 829 Standard. Beim Entwickeln, dem Testen der Software sowie der finalen Version des Produktes sind die im Pflichtenheft formulierten Qualitätsanforderungen zu berücksichtigen und nach ihrer Priorisierung umzusetzen.
So ist z.B. auf die Interoperabilität der Webanwendung in verschiedenen Browsern, die Stabilität der Anwendung im laufenden Betrieb, aber auch auf eine ausreichende Kommentierung des Codes zu achten.
Die zu testende Softwareanwendung setzt sich zusammen aus der Simulation des Unfalls, der Weboberfläche und der Unfallatenverarbeitung. Serverseitig sind die Entgegennahme, die Bereitstellung und die Verarbeitung der Unfalldaten zu prüfen. Es ist zu prüfen, ob ISANs korrekt entgegengenommen und die benötigten Daten korrekt verarbeitet und gespeichert werden und ob im Fehlerfall das System entsprechend reagiert.
Des Weiteren ist zu testen, ob die Simulation des Unfalls mit den ermittelten Daten korrekt funktioniert und als Ergebnis die simulierten 3D-Fahrzeugmodelle, sowie einige Basisdaten zu den Fahrzeugen zurückliefert und der Webanwendung bereitstellt. Die Clientanwendung ist in Bezug auf ihren definierten Funktionsumfang und ihre Nutzerfreundlichkeit zu testen. Hierbei sind die Unfallsuche, sowie die richtige Anzeige der Daten und Fahrzeuge auf der Karte zu Überprüfen. Des Weiteren muss die Webseitenlogik korrekt funktionieren.
Diese Ziele sollen mit Hilfe der Abnahme-, der Integrations- und den Unit-Test gewährleistet werden, sodass die Softwareanwendung bei Fertigstellung die Anforderungen an Funktionalität, Bedienbarkeit, Verständlichkeit, Zuverlässigkeit und Qualität erfüllt.

2 Testplan

Der Testplan ist das zentrale Dokument der Qualitätssicherung und wird daher frühzeitig erstellt. Hier wird Umfang und Vorgehensweise der Qualitätssicherung beschrieben. Außerdem werden Testgegenstände und deren zu testenden Eigenschaften bzw. Funktionen identifiziert. Ferner werden die durchzuführenden Maßnahmen definiert.

2.1 Zu testende Komponenten

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.

2.2 Zu testende Funktionen/Merkmale

Folgende Funktionen sind zu testen:

- **F1** Auswertung einer ISAN
- **F2** Zusammengehörende Unfälle werden gruppiert
- **F3** Simulation eines Unfalls
- **F4** Visualisierung eines Unfalls auf der Karte
- **F5** Suchen eines Unfalls
- **F6** Löschen eines Unfalls
- **F7** Aufrufen der Rettungskarte
- **F8** Anzeige der Fahrzeuginformationen

2.3 Nicht zu testende Funktionen

Die von den Tests ausgenommenen Funktionen umfassen das Server-Betriebssystem Windows 10 64-Bit sowie die Simulationsumgebung BeamNG. Des Weiteren werden auch das verwendete Angular-Framework und die Pakete der Programmiersprachen JavaScript und Python von den Tests ausgeschlossen. Da die genannten Produkte bereits etabliert und validiert auf dem Markt sind, ist eine weitere Überprüfung als entbehrlich anzusehen.
Des Weiteren ist die Funktion **F9** vom Testen ausgeschlossen, da sie gemäß dem technischen Entwurf im Kapitel 8 nicht entwickelt wurde. Die genauen Gründe dafür wurden dort erläutert.

2.4 Vorgehen

In diesem Abschnitt werden die allgemeinen Vorgehensweisen für die einzelnen zu testenden Funktionen und Komponenten beschrieben.

2.4.1 Komponententest

Beim Komponententest wird alle in 2.1 genannte Komponente isoliert voneinander getestet. Dort werden die einzelnen Testfälle getestet und die Ergebnisse auch protokoliert. Dadurch kann Fehlerursachen genauer nachvollgezogen werden.

2.4.2 Integrationstest

Nach dem Erfolg des Komponententests werden die Komponenten vom jeweiligen Teil der Anwendung integriert und zusammengetestet. Ziel ist es, dass sie korrekt miteinander interagieren und die gewünschten Funktionalitäten bereitstellen. In diesem Fall wird die Komponente von der Webanwendung und der Simulation integriert und zusammengetestet.

2.4.3 Systemtest

Beim Systemtest wird das vollständige und integrierte System auf ihre Leistung, Funktionalität, und Konformität mit der Anforderungen überprüft und getestet. Hier werden die Webanwendung und die Simulation integriert und getestet, ob das gesamte System den erwarteten Spezifikationen entspricht.

2.4.4 Abnahmetest

Die gesamte Anwendung wird von dem Auftraggeber überprüft, um sicherzustellen, dass sie ihren Anforderungen, Erwartungen und den vereinbarten Kriterien basierend von der Pflichtenheft entspricht. Nach dem Erfolg von Abnahmetest folgt dann auch der rechtliche Abschluss des Vertrags und die Übergabe des Produkts.

2.5 Testumgebung

Für die Testphase des Backends wird ein Server-Rechner als Testumgebung verwendet. Dieser Schritt ist erforderlich, um sicherzustellen, dass das Backend einwandfrei funktioniert, bevor es in einer produktiven Umgebung implementiert wird.

Für das Frontend werden die Funktionalitäten in der Regel visuell getestet und dabei das folgende Developer-Tool genutzt: Angular DevTools. Diese Tools bieten den Entwicklern und Testern die Möglichkeit, das Frontend interaktiv zu inspizieren, zu überwachen und zu debuggen.

3 Abnahmetest

In diesem Kapitel werden die Abnahmetests genauer spezifiziert. Mit Hilfe der Abnahmetests wird überprüft, ob die im Pflichtenheft festgelegten Anforderungen an die zu entwickelnde Software, vollständig und korrekt erfüllt wurden. Es wird festgelegt welche Anforderung mit welchem Testverfahren auf ihrer Korrektheit überprüft werden.

3.1 Zu testende Anforderungen

In diesem Abschnitt werden alle zu testenden Anforderungen aufgeführt

Nr	**Anforderung**	**Testfälle**	**Kommentar**
1	Funktion <F1>	<T1>, <T2>	Wird die ISAN korrekt ausgewertet?
2	Funktion <F2>	<T3>, <T4>	Werden zusammengehörende Unfälle gruppiert?
3	Funktion <F3>	<T5>	Werden Unfälle korrekt simuliert?
4	Funktion <F4>	<T6>	Werden die Daten des Unfalls korrekt übertragen und auf der Karte angezeigt?
5	Funktion <F5>	<T7>	Kann der Nutzer auf einen Unfall suchen?
6	Funktion <F6>	<T8>	Werden zu alte Unfälle vom Server entfernt?
7	Funktion <F7>	<T9>	Wird die richtige Rettungskarte geöffnet?
8	Funktion <F8>	<T10>	Werden die Informationen des Fahrzeugs angezeigt?

3.2 Testverfahren

Die Tests werden im Black-Box-Verfahren durchgeführt, indem die Ausgabe einer Funktion mit der erwarteten Ausgabe verglichen wird. Dabei werden die Tests manuell vom Entwickler durchgeführt, indem er die Eingaben selbst eingibt. Die Ausgaben werden entweder mithilfe von Tools wie Debugger oder visuell bewertet.

3.2.1 Testskripte

Wir verzichten auf den Einsatz von Testskripten, da der Arbeitsaufwand zur Erstellung der Testumgebung und der Testfälle in keinem Verhältnis zur Komplexität der notwendigen Test steht. Stattdessen testen wir die Funktionen Schritt für Schritt, indem wir sie ausführen und den Zustand der Variablen mithilfe von Logging und/oder Debugger-Tools überprüfen, um sicherzustellen, dass die Zustände korrekt sind. GUI-Funktionen werden visuell überprüft.

3.3 Testfälle

Hier werden die Testfälle im Detail beschrieben, einschließlich der Vorbedingungen, der einzelnen Schritte und der Pass-/Fail-Kriterien, sowie anderer relevanter Informationen. Dies ermöglicht eine genaue Beschreibung und Durchführung der Tests.

3.3.1 Testfall $\langle T1 \rangle$ - Auswerten einer ISAN (mit korrektem Format)

Ziel

Hier wird getestet, ob die Anwendung die richtigen Daten aus einer ISAN mit korrektem Format extrahiert.

Objekte/Methoden/Funktionen

Funktion <F1>

Pass/Fail Kriterien

Der Test ist erfolgreich, wenn die Anwendung die korrekten Daten aus einer gültigen ISAN extrahiert und erfolgreich einen neuen Unfall auf dem Server speichert.

Vorbedingung

Eine ISAN wurde an das System übergeben.

Einzelschritte

1. Der Server empfängt eine ISAN.
2. Der Server wertet die ISAN erfolgreich aus.
3. Das REST-API gibt eine erfolgreiche Response zurück.
4. Es wird dann getestet, ob die extrahierten Daten zu den Daten in der ISAN passen, d.h. die Werte und Datentypen sind korrekt.

Beobachtungen / Log / Umgebung

Ein neuer Unfall muss im Anschluss erzeugt werden.

Besonderheiten Es müssen Dummy-ISAN's vorliegen, welche ein korrektes Format haben und dessen Daten bekannt sind.

Abhängigkeiten

3.3.2 Testfall $\langle T2 \rangle$ - Auswerten einer ISAN (mit ungültigem Format)

Ziel

Hier wird getestet, ob die Anwendung eine ISAN mit falschem Format erkennt.

Objekte/Methoden/Funktionen

Funktion <F1>

Pass/Fail Kriterien

Der Test ist erfolgreich, ob die Anwendung eine ISAN mit falschem Format nicht akzeptiert bzw. entsprechend reagiert.

Vorbedingung

Eine ISAN wurde an das System übergeben.

Einzelschritte

1. Der Server empfängt eine neue ISAN.
2. Der Server kann die ISAN nicht auswerten.
3. Das REST-API gibt eine fehlgeschlagene Response zurück.

Beobachtungen / Log / Umgebung

Ein neuer Unfall darf nicht erzeugt werden. Es folgen keine weiteren Schritte auf dem Web-Server nach Übermittlung der fehlgeschlagenen Response.

Besonderheiten Es müssen Dummy-ISAN's vorliegen, welche ein falsches Format haben.

Abhängigkeiten

3.3.3 Testfall $\langle T3 \rangle$ - Gruppierung zusammengehörender Unfälle

Ziel

Hier wird getestet, ob die Anwendung mehrere ISAN's, welche zeitlich und geografisch ähnliche Informationen führen, als einen einzigen Unfall identifizieren kann.

Objekte/Methoden/Funktionen

Funktion <F2>

Pass/Fail Kriterien

Der Test ist erfolgreich, wenn die Anwendung Unfälle, die innerhalb eines bestimmten Zeitraums und räumlich nahe zueinander stattfinden, korrekt als einen einzigen Unfall erkennt. Dies stellt sicher, dass nur ein Unfall erzeugt wird, der alle betroffenen Fahrzeuge enthält.

Vorbedingung

Mindestens Ein Unfall ist auf dem Server gespeichert.

Einzelschritte

1. Der Server empfängt eine neue ISAN (dieser Unfall gehört zu mindestens einem Unfall der schon gespeichert ist).
2. Der Server wertet die ISAN aus.
3. Der Server stellt fest, dass dieser Unfall zu mindestens einem bereits gespeicherten Unfall gehört.
4. Der Server gibt eine erfolgreiche Response zusammen mit der Unfall-ID zurück.

Beobachtungen / Log / Umgebung

Der Unfall wird mit einem bereits gespeicherten Unfall gruppiert.

Besonderheiten Es liegen Dummy-ISAN'S vor, welche von mehreren Fahrzeugen stammen, die zu einem Unfall gehören.

Abhängigkeiten

3.3.1

3.3.4 Testfall $\langle T4 \rangle$ - Unterscheidung auseinandergehörender Unfälle

Ziel

In diesem Test wird überprüft, ob die Anwendung richtig erkennt, dass nicht zusammengehörende Unfälle nicht gruppiert werden sollten.

Objekte/Methoden/Funktionen

Funktion <F2>

Pass/Fail Kriterien

Der Test ist erfolgreich, wenn die Anwendung Unfälle, die sich physikalisch weit voneinander entfernt befinden oder die zeitlich weit auseinander liegen, nicht in einen neuen Unfall gruppiert.

Vorbedingung

Mindestens Ein Unfall ist auf dem Server gespeichert.

Einzelschritte

1. Der Server empfängt eine neue ISAN (dieser Unfall gehört zu keinem anderen Unfall im System).
2. Der Server wertet die ISAN aus.
3. Der Server prüft, ob dieser Unfall zu mindestens einem bereits gespeicherten Unfall gehört.

Beobachtungen / Log / Umgebung

Ein neuer Unfall muss erzeugt werden.

Besonderheiten Es liegen Dummy-ISAN'S vor, welche von mehreren Fahrzeugen stammen, die nicht zu einem Unfall gehören.

Abhängigkeiten

3.3.1

3.3.5 Testfall $\langle T5 \rangle$ - Automatische Simulation eines Unfalls

Ziel

In diesem Test wird überprüft, ob der Simulations-Server vollautomatisch eine Simulation durchführen und die Ergebnisse liefern kann. Die Kommunikation findet dabei nur über die REST-API zwischen Web-Server und Simulations-Server statt.

Objekte/Methoden/Funktionen

Funktion <F3>

Pass/Fail Kriterien

Der Test ist erfolgreich, wenn der Simulations-Server nach eingegangener GET-Request eine Simulation startet und nach Beendigung die Ergebnisse der Simulation zurück an den Web-Server als Response übermittelt.

Vorbedingung

Auf dem Simulations-Server wird gerade keine Simulation ausgeführt.

Einzelschritte

1. Der Web-Server fragt den Simulations-Server mittels einer GET-Request nach einer Simulation mit den übermittelten Daten.
2. Der Simulations-Server empfängt die Daten und startet eine Simulation.
3. Wenn die Simulation beendet ist, liest der Simulations-Server die Ergebnisse aus.
4. Die Ergebnisse werden dem Web-Server als Response zurückgegeben.

Beobachtungen / Log / Umgebung

Besonderheiten

Die Zeit die der Simulations-Server für die Response braucht kann variieren und sollte nicht zum Fehlschlag dieses Tests führen.

Abhängigkeiten

3.3.1

3.3.6 Testfall $\langle T6 \rangle$ - Unfall visualisieren

Ziel

In diesem Test wird überprüft, ob ein Unfall korrekt auf der Karte visualisiert wird.

Objekte/Methoden/Funktionen

Funktion <F4>

Pass/Fail Kriterien

Der Test gilt als erfolgreich, wenn die Webanwendung die Daten eines Unfalls vom Server empfangen kann und den Unfall entsprechend korrekt in der Webanwendung darstellt, wobei die angezeigten Informationen den gegebenen Daten entsprechen.

Vorbedingung

Mindestens ein Unfall, dessen ID bekannt ist, ist bereits auf dem Server gespeichert.

Einzelschritte

1. Die ID eines Unfalls wird in das Suchfeld eingegeben..
2. Die Webanwendung fordert die Daten des Unfalls und der Server schickt die zurück.
3. Die Karte wird geladen.
4. Auf der Karte befindet sich mindestens ein 3D-Modell eines beschädigten Fahrzeugs an der geografischen Stelle, welche ursprünglich mit der ISAN übermittelt wurde.

Beobachtungen / Log / Umgebung

Besonderheiten

Abhängigkeiten

3.3.1, 3.3.3, 3.3.4 3.3.5, 3.3.7

3.3.7 Testfall $\langle T7 \rangle$ - Suchen eines Unfalls

Ziel

In diesem Test wird überprüft, ob die Anwendung einen Unfall anhand seiner ID finden kann.

Objekte/Methoden/Funktionen

Funktion <F5>

Pass/Fail Kriterien

Der Test gilt als erfolgreich, wenn der Nutzer die ID eines Unfalls eingeben kann und der Server den passenden Unfall finden kann und die Daten dieses Unfalls erfolgreich an die Webanwendung übermittelt werden können.

Vorbedingung

Mindestens ein Unfall, dessen ID bekannt ist, ist bereits auf dem Server gespeichert.

Einzelschritte

1. Die ID eines Unfalls wird in das Suchfeld eingegeben..
2. Die Webanwendung fordert die Daten des Unfalls
3. Der Server sucht nach einem Unfall, dessen ID der angeforderten ID entspricht
4. Der Server schickt die Daten des Unfall zurück, falls die ID existiert und zu einem Unfall gehört, sonst schickt er eine fehlgeschlagene Response.

Beobachtungen / Log / Umgebung

Besonderheiten

Abhängigkeiten

3.3.1, 3.3.3, 3.3.4, 3.3.5

3.3.8 Testfall $\langle T8 \rangle$ - Löschen eines Unfalls

Ziel

In diesem Test wird überprüft, ob die Anwendung einen Unfall nach einer vorab-definierten Zeit löscht.

Objekte/Methoden/Funktionen

Funktion <F6>

Pass/Fail Kriterien

Der Test gilt als erfolgreich, wenn alle zu löschenden Unfälle erfolgreich vom Server entfernt werden. Alle Unfälle, welche die vorab-definierte Lebenszeit noch nicht überschritten haben sollten nicht gelöscht werden.

Vorbedingung

Auf dem Server muss mindestens ein Unfall existieren, welcher gelöscht werden soll und einer, welcher nicht gelöscht werden soll.

Einzelschritte

1. Alle Unfälle werden überprüft, ob ihre Lebenszeit abgelaufen ist.
2. Wird ein solcher Unfall gefunden, für den das zutrifft, wird er gelöscht.

Beobachtungen / Log / Umgebung

Besonderheiten

Abhängigkeiten

3.3.1, 3.3.3, 3.3.4

3.3.9 Testfall $\langle T9 \rangle$ - Rettungskarte aufrufen

Ziel

Der Test überprüft, ob die Rettungskarte eines Autos ordnungsgemäß geöffnet wird.

Objekte/Methoden/Funktionen

Funktion <F7>

Pass/Fail Kriterien

Wird die richtige Rettungskarte geöffnet gilt der Test als bestanden, sonst nicht.

Vorbedingung

Es muss ein Unfall mit mindestens einem Auto auf dem Server existieren.

Einzelschritte

1. Suche den Unfall
2. Richte die Karte auf ein Auto
3. Klicke auf den Button zum Öffnen der Rettungskarte eines Autos

Beobachtungen / Log / Umgebung

Besonderheiten

Abhängigkeiten

3.3.7 3.3.6 3.3.4 3.3.3 3.3.1

3.3.10 Testfall $\langle T10 \rangle$ - Fenster mit Zusatzinformationen zu einem Fahrzeug aufrufen

Ziel

Der Test überprüft, ob das Fenster, welches zusätzliche Informationen zu einem Fahrzeug angibt, angezeigt wird, wenn sich der Mauszeiger über dem Fahrzeug befindet.

Objekte/Methoden/Funktionen

Funktion <F8>

Pass/Fail Kriterien

Der Test gilt als bestanden, falls sich das Fenster korrekt öffnet und schließt und es auch die korrekten Informationen anzeigt.

Vorbedingung

Es muss ein Unfall mit mindestens einem Auto auf dem Server existieren und die Karte mit diesem Unfall muss geöffnet sein.

Einzelschritte

1. Richte die Karte auf ein Auto.
2. Fahre mit dem Mauszeiger über ein Auto.
3. Ein Fenster öffnet sich.
4. Das Fenster sollte die zusätzlichen Informationen korrekt anzeigen.
5. Nehme den Mauszeiger von dem Auto weg.
6. Das Fenster sollte sich schließen.

Beobachtungen / Log / Umgebung

Besonderheiten

Abhängigkeiten

3.3.7, 3.3.6, 3.3.5, 3.3.4, 3.3.3, 3.3.1

4 Integrationstest

In diesem Kapitel wird die Vorgehensweise beim Integrationstest erläutert. Es werden in den folgenden Abschnitten die Testziele, die zu testenden Komponenten, das Testverfahren, die Testskripte und die konkreten Testfälle erläutert.

4.1 Zu testende Komponenten

Nr	Komponenten	Testfälle	Kommentar
1	<T12> Simulation	4.3.1	
2	<T13> Webserver, Unfallserver	4.3.2	

4.2 Testverfahren

Die Tests werden im Black-Box-Verfahren durchgeführt, indem die Ausgabe einer Funktion mit der erwarteten Ausgabe verglichen wird. Dabei werden die Tests manuell vom Entwickler durchgeführt, indem er die Eingaben selbst eingibt. Die Ausgaben werden entweder mithilfe von Tools wie Debugger oder visuell bewertet. ...

4.2.1 Testskripte

Für die Integrationstests wird Karma und Jasmine (Angular) verwendet.

4.3 Testfälle

Hier ist die Liste der Integrationstests: Die Integration zwischen dem Unfallservice und dem Webserver wird teilweise getestet, ebenso wie die Integration zwischen dem Webserver und dem Simulations-Server.

4.3.1 Testfall $\langle T12 \rangle$ - Komponente Simulation + Datenauswertung

Ziel Dieser Test überprüft, ob der Webserver und der Simulations-Server reibungslos zusammenarbeiten können, d.h. dass der Webserver Anfragen für Simulationen an den Simulations-Server gibt und dieser auf diese mit einem Mesh antwortet.

Pass/Fail Kriterien Der Test ist erfolgreich, wenn die zuvor beschrieben Kommunikation reibungslos funktioniert.

Vorbedingung Der Webserver und der Simulations-Server müssen parallel auf dem selben Rechner laufen. Es müssen Beispiel-Anfragen vorliegen.

Einzelschritte

1. Der Web-Server fragt den Simulations-Server mittels einer GET-Request nach einer Simulation mit den übermittelten Daten.
2. Der Simulations-Server empfängt die Daten und startet eine Simulation.
3. Wenn die Simulation beendet ist, liest der Simulations-Server die Ergebnisse aus.
4. Die Ergebnisse werden dem Web-Server als Response zurückgegeben.

Besonderheiten Dieser Test wird manuell ausgeführt. Dieser Test orientiert sich an dem Test 3.3.5, welche bereits in den Abnahmetestspezifikationen aufgeführt wurde. Da es sich dabei um einen Integrationstest handelt, haben wir ihn hier noch einmal aufgeführt.

Abhängigkeiten 3.3.5 5.3.1

4.3.2 Testfall $\langle T13 \rangle$ - Komponente Webserver+Unfallserver

Ziel Testen, ob die Daten vom Webserver erfolgreich an den Unfallservice übertragen und verarbeitet werden.

Objekte/Methoden/Funktionen getAccidentById

Pass/Fail Kriterien **Pass:** Der Unfallserver verarbeitet und speichert die empfangenen Daten vom Webserver korrekt

Fail: Der Unfallserver verursacht Fehler, wirft Ausnahmen oder verarbeitet die empfangenen Daten nicht korrekt

Vorbedingung Es existiert ein Unfall mit derselben ID im Webserver.

Einzelschritte

1. Das Projektverzeichnis öffnen
2. ng test–sinclude=src/app/accident.service.integration.spec.ts.

Beobachtungen / Log / Umgebung Es wird im Browser angezeigt, ob der Test erfolgreich ist

Besonderheiten keine

Abhängigkeiten keine

5 Unit-Tests

Die Funktionalität unserer Komponente wird in isolierten Unit-Tests getestet. Dabei werden die einzelnen Methoden genau überprüft, um anschließend mit Integrationstests und Abnahmetests fortzufahren.

5.1 Zu testende Komponenten

Nr	Komponenten	Testfälle	Kommentar
1	<C40> Simulation-Server	5.3.1	Klasse Simulator
2	<C40> Simulation-Server	5.3.2	Klasse Simulation
3	<C20> Unfallservice	5.3.3	Fetch accident
4	<C20> Unfallservice	5.3.4	Fetch cached accident
5	<C20> Unfallservice	5.3.5	Reset accident
5	<C40> GUI	5.3.6	Load accident on homepage initialisation
5	<C40> GUI	5.3.7	Fetch accident by giving ID
5	<C30> Webserver	5.3.8	Get accident by ID
5	<C30> Webserver	5.3.9	Delete expired accidents
5	<C30> Webserver	5.3.10	Create ISAN
5	<C30> Webserver	5.3.11	Gruppierung der Unfälle
5	<C40> GUI	5.3.12	Visualize Accident

5.2 Testverfahren

Black-Box-Tests

5.2.1 Testskripte

Für die Unittest wird unittest (Python-Module), Karma und Jasmine (Angular) und Jest(Javascript) verwendet.

5.3 Testfälle

Im Folgenden werden alle Testfälle aufgeführt und deren Ziele erläutert. Nachfolgend finden Sie eine Liste der Testfälle für die Unittests. Zunächst wird das Ziel jedes einzelnen Tests erläutert. Dazu werden die betroffenen Funktionen aufgeführt und die erwarteten Ergebnisse ("Passöder "Fail") für den jeweiligen Test angegeben. Anschließend wird der Ablauf des spezifischen Unittests beschrieben und die Umgebung, in der das Testergebnis beobachtet werden kann, erläutert.

5.3.1 Testfall $\langle T20 \rangle$ - Klasse Simulator

Ziel Teste ob eine Simulation ausgeführt werden kann und ob ein 3D-Objekt im korrekten Format ausgegeben wird.

Objekte/Methoden/Funktionen

Objekt: Simulator

Methode: Simulator.run_scenario()

Pass/Fail Kriterien Der Test ist erfolgreich, wenn die Simulation beendet wird, ein 3D-Objekt zurück gibt und die Validierung dieses 3D-Objekts keine Fehler ausgibt.

Vorbedingung Das Objekt Simulator muss instanziert sein.

Einzelschritte

1. Rufe Simulator.run_scenario() auf
2. Überprüfe ob der Aufruf ein Mesh zurück gibt
3. Validiere das Mesh

5.3.2 Testfall $\langle T21 \rangle$ - Klasse Simulation

Ziel Teste ob die REST-API des Simulations-Servers korrekt funktioniert.

Objekte/Methoden/Funktionen

Objekt: Simulation

Methode: Simulator.get()

Pass/Fail Kriterien Der Test ist erfolgreich, wenn auf eine geeignete Get-Request eine erfolgreiche Response folgt, welche das Mesh beeinhaltet.

Vorbedingung Die REST-API muss laufen. Es müssen korrekte Beispiel-Anfragen vorliegen.

Einzelschritte

1. Benutze curl für die Anfragen:
   ´curl -X GET 'http://128.0.0.1:5000/simulation?force
   =[FILL]speed=[FILL]hsn=[FILL]tsn=[FILL]'´
2. Überprüfe ob nach geeigneter Zeit eine Response mit dem Mesh zurückkommt.

Bemerkung Dieser Test wird manuell ausgeführt.

5.3.3 Testfall $\langle T22 \rangle$ - Fetch accident

Ziel

Teste Sie, ob unser Unfallservice einen Unfall durch HTTP-Anfragen empfangen kann.

Objekte/Methoden/Funktionen

getAccidentByID

Pass/Fail Kriterien

Ein Simulation RESt-API-Server wird ausgeführt. Es wird dann überprüft, ob der empfangene Unfall mit dem gespeicherten Unfall übereinstimmt.

Pass: Der empfangene Unfall stimmt mit dem gespeicherten Unfall überein.

Fail: Der empfangene Unfall stimmt nicht mit dem gespeicherten Unfall überein.

Vorbedingung

Die Simulierte REST-API muss laufen.

Einzelschritte

1. Das Projektverzeichnis öffnen.
2. das Kommando 'ng test –include=src/app/accident.service.spec.ts' ausführen.

Beobachtungen / Log / Umgebung

Es wird im Browser angezeigt ob der Test erfolgreich ist

Besonderheiten

keine

Abhängigkeiten

keine

5.3.4 Testfall $\langle T23 \rangle$ - Fetch cached accident

Ziel

Teste Sie, ob unser Unfallservice einen Unfall zwischenspeichern, und später zur Verfügung stellen kann

Objekte/Methoden/Funktionen

getAccident

Pass/Fail Kriterien

Ein Simulation REST-API-Server wird ausgeführt. Es wird dann überprüft, ob der zwischengespeicherte Unfall, mit dem gespeicherten Unfall übereinstimmt.

Pass: Der zwischengespeicherte Unfall stimmt mit dem gespeicherten Unfall überein.

Fail: Der zwischengespeicherte Unfall stimmt nicht mit dem gespeicherten Unfall überein.

Vorbedingung

Die Simulierte REST-API muss laufen.

Einzelschritte

1. Das Projektverzeichnis öffnen.
2. das Kommando 'ng test –sinclude=src/app/accident.service.spec.ts' ausführen.

Beobachtungen / Log / Umgebung

Es wird im Browser angezeigt ob der Test erfolgreich ist

Besonderheiten

keine

Abhängigkeiten

5.3.3

5.3.5 Testfall $\langle T24 \rangle$ - Reset accident

Ziel

Teste, ob unser Unfallservice einen zwischengespeicherten Unfall löschen kann.

Objekte/Methoden/Funktionen

resetAccident

Pass/Fail Kriterien

Ein Simulation RESt-API-Server wird ausgeführt. Es wird dann überprüft, ob der zwischengespeicherte Unfall gelöscht werden kann.

Pass: Der zwischengespeicherte Unfall wird gelöscht.

Fail: Der zwischengespeicherte Unfall wird nicht gelöscht.

Vorbedingung

Die Simulierte REST-API muss laufen.

Einzelschritte

1. Das Projektverzeichnis öffnen.
2. das Kommando 'ng test –sinclude=src/app/accident.service.spec.ts' ausführen.

Beobachtungen / Log / Umgebung

Es wird im Browser angezeigt ob der Test erfolgreich ist

Besonderheiten

keine

Abhängigkeiten

5.3.3

5.3.6 Testfall $\langle T25 \rangle$ - Load accident on home page initialisation

Ziel

Teste, ob unsere Unfalldaten während einer Initialisierung geladen werden.

Objekte/Methoden/Funktionen

ngOnInit

Pass/Fail Kriterien

Pass: Die Komponente ruft den Unfallservice an und speichert die Daten korrekt.

Fail: Die Daten werden nicht korrekt gespeichert .

Vorbedingung

Ein mockaccident Datei muss vorhanden sein.

Einzelschritte

1. Das Projektverzeichnis öffnen.
2. das Kommando 'ng test –sinclude=src/app/accident-page/accident-page.component.spec.ts' ausführen.

Beobachtungen / Log / Umgebung

Es wird im Browser angezeigt ob der Test erfolgreich ist

Besonderheiten

keine

Abhängigkeiten

keine

5.3.7 Testfall $\langle T26 \rangle$ - Fetch accident by giving ID

Ziel

Teste, ob der Unfallservice benachrichtigt wird und ob eine Weiterleitung zur “Accident-Page”erfolgt, wenn eine Unfall-ID eingegeben wird.

Objekte/Methoden/Funktionen

onSubmitSearch

Pass/Fail Kriterien

Pass: Die Methode "getAccidentbyID"des Unfallservices wird aufgerufen, und der Browser wird zum Unfall weitergeleitet.

Fail: Die Pfade ändern sich nicht, oder der Unfallservice wird nicht aufgerufen.

Vorbedingung

Ein mockaccident Datei muss vorhanden sein.

Einzelschritte

1. Das Projektverzeichnis öffnen.
2. das Kommando ’ng test –sinclude=src/app/home-page/home-page.component.spec.ts’ ausführen.

Beobachtungen / Log / Umgebung

Es wird im Browser angezeigt ob der Test erfolgreich ist

Besonderheiten

keine

Abhängigkeiten

keine

5.3.8 Testfall $\langle T27 \rangle$ - Get Accident by ID

Ziel

Testen, ob der Server die richtige JSON-Datei der angegebenen ID bekommt.
Testen, ob das Unfallobjekt korrekt aus der Datei extrahiert wird

Objekte/Methoden/Funktionen

getAccidentByID

Pass/Fail Kriterien

Pass: Der erhaltene Unfall stimmt mit dem gespeicherten Unfall überein.
Fail: Der erhaltene Unfall stimmt nicht mit dem gespeicherten Unfall überein.

Vorbedingung

Die Simulierte REST-API muss laufen.

Einzelschritte

1. Das Projektverzeichnis öffnen.
2. Das Kommando ' npm test ' ausführen.

Beobachtungen / Log / Umgebung

Es wird im Browser angezeigt ob der Test erfolgreich ist

Besonderheiten

keine

Abhängigkeiten

keine

5.3.9 Testfall $\langle T28 \rangle$ - Delete expired accidents

Ziel

Teste, ob die abgelaufenen Unfälle nach einem bestimmten Zeitraum gelöscht werden.

Objekte/Methoden/Funktionen

deleteExpiredAccidents

Pass/Fail Kriterien

Pass: Der Unfall wird nach 5 Stunden gelöscht .

Fail: Der Unfall wird nicht gelöscht .

Vorbedingung

Der Unfall muss erfolgreich gespeichert.

Einzelschritte

1. Das Projektverzeichnis öffnen.
2. Das Kommando ' npm test ' ausführen.

Beobachtungen / Log / Umgebung

Es wird im Browser angezeigt, ob der Test erfolgreich ist

Besonderheiten

keine

Abhängigkeiten

keine

5.3.10 Testfall $\langle T29 \rangle$ - Create ISAN

Ziel

Testen, ob ein neuer Unfall erfolgreich mit einem validen ISAN erstellt werden kann.

Objekte/Methoden/Funktionen

createISAN

Pass/Fail Kriterien

Pass: Der Statuscode ist 201 und die Erfolgsmeldung wird zurückgegeben.

Fail: Der Statuscode ist nicht 201 oder die Erfolgsmeldung wird nicht zurückgegeben.

Vorbedingung

Die ISAN des Unfalls muss gültig sein.

Einzelschritte

1. Einen HTTP POST-Request an den Endpunkt /accidentsßenden mit einem validen ISAN.
2. Überprüfe ob nach geeigneter Zeit eine Erfolgsmeldung zurückkommt.

Beobachtungen / Log / Umgebung

Es wird im Browser angezeigt, ob der Test erfolgreich ist

Besonderheiten

keine

Abhängigkeiten

keine

5.3.11 Testfall $\langle T30 \rangle$ - Gruppierung der Unfällen

Ziel

Teste, ob ein Unfall, ob ein Unfall, der zeitlich und geografisch ähnlich zu anderen Unfällen ist, korrekt gruppiert werden kann.

Objekte/Methoden/Funktionen

searchAndGroupAccidents

Pass/Fail Kriterien

Pass: Der Unfall wird korrekt gruppiert.

Fail: Der Unfall wird nicht korrekt gruppiert.

Vorbedingung

Es existiert zumindest einen Unfall, der zeitlich und geografisch ähnlich zu dem gegebenen Unfall ist.

Einzelschritte

1. Rufe die Funktion mit der Suchanfrage und den Gruppierungskriterien auf.
2. Überprüfe, ob die Funktion die richtige Gruppe von Unfällen gemäß den Such- und Gruppierungskriterien liefern.

Beobachtungen / Log / Umgebung

Es wird im Browser angezeigt, ob der Test erfolgreich ist

Besonderheiten

keine

Abhängigkeiten

keine

5.3.12 Testfall $\langle T31 \rangle$ - Visualize Accident

Ziel

Dieser Test arbeitet mit einem hinterlegten Dummy-Unfall. Dieser Test überprüft, ob die Webseite die Daten des Unfalls aus der .json Datei lesen kann. Es wird überprüft, ob die Meshes der Unfallfahrzeuge korrekt auf der Karte angezeigt werden und die Zusatzinformationen zu den Fahrzeugen geladen werden und durch Auswahl des Informationssymbols angezeigt werden. Des Weiteren funktioniert die Navigation um den Unfall über die Tasten auf der Karte.

Objekte/Methoden/Funktionen

initMap

Pass/Fail Kriterien

Pass: Die Webanwendung lädt die Google-Maps Karte. Auf dieser Karte werden die zwei Unfallfahrzeuge angezeigt. An der Position eines jeden Unfallfahrzeuges wird ein Info Marker angezeigt. Bei Auswahl eines Marker, werden Zusatzinformationen angezeigt. Das Infofenster lässt sich schließen. Mit Hilfe der Navigationstasten auf der Karte kann um den Unfall im 3D-Raum navigiert werden.
In der Console im Browser ist sichtbar, dass "Mesh 0ünd "Mesh 1"geladen wurden.
Fail: Die Modelle werden nicht geladen oder die Navigation funktioniert nicht oder die Zusatzinformationen werden nach Auswahl des entsprechenden Info-Symbol nicht angezeigt.

Vorbedingung

Es existiert ein Dummy-Unfall mit zwei Unfallfahrzeugen gespeichert in der db.json Datei mit der gültigen Unfall-ID 22. Es ist eine Internetverbindung notwendig.

Einzelschritte

1. Webanwendung mittels "ng serveßtarten und "json-server –watch db.jsonßtarten
2. Webseite aufrufen über "localhost:4200/äufrufen
3. Unfall-ID 22 in das Suchfeld eingeben und Suche starten
4. Es wird die Seite mit der Karte geladen
5. Überprüfen in der Console des Browser, ob Meshes geladen wurden
6. Überprüfen ob die zwie Fahrzeuge geladen wurden
7. Überprüfen, ob zwei Info-Marker angezeigt werden und ob bei deren Auswahl die jeweiligen Zusatzinformationen angezeigt werden.
8. Überprüfen, ob die 3D-Navigation um den Unfall funktioniert.

Beobachtungen / Log / Umgebung

Überprüfung erfolgt visuell im Browser und über die Console im Browser. In der Console werden die Arrays mesh 0 und mesh 1 angezeigt.

Besonderheiten

keine

Abhängigkeiten

keine

6 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.

BeamNG ist eine physikbasierte Fahrzeug-Simulation, die es Spielern ermöglicht, realistische Crash- und Fahrerlebnisse in einer offenen Spielwelt zu erleben.

IEEE 829 Standard gilt als der bekannteste Standard für Software-Testdokumentationen. Der Standard definiert eine Menge von Testdokumenten und beschreibt deren Inhalte.

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.

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

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