Kollaborative Annotationen und die Persistierung durch Snapshots in einer 3D Software Visualisierung

Philipp Wulff

Bachelorarbeit
9. September 2024

Software Engineering Group
Institut für Informatik
Christian-Albrechts-Universität zu Kiel

Betreut durch
Prof. Dr. Wilhelm Hasselbring
Malte Hansen, M.Sc.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Weiterhin erkläre ich, dass die digitale Fassung dieser Arbeit, die dem Prüfungsamt per E-Mail zugegangen ist, der vorliegenden schriftlichen Fassung entspricht.

Kiel,

Zusammenfassung

Entwickler verbringen viel Zeit mit dem Verstehen von existierender Software, weshalb Software-Verständnis wichtig für die Softwareentwicklung ist. Zur Unterstützung für Entwickler gibt es für Softwaresysteme Visualisierungswerkzeuge, die bei der Erarbeitung dieses Verständnisses helfen. Eines dieser Werkzeuge ist die webbasierte Open-Source Anwendung ExplorViz, welche die 3D-City Metapher umsetzt und das kollaborative Arbeiten in ihr unterstützt. In diesem können Entwickler sich Softwaresysteme und deren Kommunikation anschauen. Allerdings ist es innerhalb des Systems nicht möglich Informationen zu notieren und diese mit anderen zu teilen.
In dieser Arbeit präsentieren wir einen Ansatz für die Erweiterung von ExplorViz durch eine Annotationsfunktion. Die Annotationen können in der Ansicht von ExplorViz frei oder zugeordnet zu Objekten erstellt werden. Es ist möglich, die Annotationen frei zu bewegen, zu minimieren, und zu löschen. Außerdem sind die Annotationen in den Kollaborationsmodus von ExplorViz integriert, so dass es möglich ist, Annotationen mit mehreren Teilnehmern zu teilen und diese für alle zu bearbeiten. Zudem präsentieren wir einen Ansatz für die Persistierung der Daten einer Snapshotfunktion und einer Git-Hosting-Anbindung. Dieser beinhaltet die Speicherung der Daten in einer Mongo-Datenbank und eine REST-Schnittstelle, um mit dem Frontend von ExplorViz zu kommunizieren.
In einer Usability-Studie haben wir Rückmeldungen zu der Benutzbarkeit und dem Design unseres Ansatzes erhalten. Dafür haben 14 Probanden einen Online-Fragebogen ausgefüllt, der ihnen Aufgaben zu den implementierten Funktionen stellte. Diese Aufgaben haben die Probanden in ExplorViz bearbeitet. Im Anschluss an diese sollten Fragen zu der Benutzung und dem Design der Funktionen beantwortet werden. Die Ergebnisse der Evaluation zeigen, dass die implementierten Funktionen sinnvoll in ExplorViz verwendet werden können und gut benutzbar sind. Die Rückmeldungen ergaben außerdem mögliche Verbesserungen und Veränderungen für den implementierten Ansatz. Diese Vorschläge haben wir genutzt, um einen Ausblick auf die zukünftige Arbeit an den Funktionen zu geben. Hierzu gehört die Erstellung eines Übersichtsmenüs für die Annotationen, über das minimierte Annotationen wieder geöffnet werden können und Informationen zu den Annotationen angezeigt werden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1. Motivation

Entwickler verbringen viel Zeit mit dem Verstehen von existierender Software [Xia u. a. 2018]. Deshalb ist das Verständnis von Software ein wichtiger Prozess in der Softwareentwicklung [Hasselbring u. a. 2020]. Um diesen Prozess zu unterstützen gibt es Visualisierungswerkzeuge, die ein Softwaresystem analysieren und darstellen [Wettel u. a. 2011]. Ein Ansatz, den es hierfür gibt, ist der der 3D Cody Cities bzw. der City Metapher [Wettel und Lanza 2007]. Hierbei werden Softwaresysteme in einer dreidimensionalen Darstellung aufgebaut. In dieser werden Pakete als Gebäudeblöcke dargestellt, welche je nach Anwendung in der Größe abhängig von bestimmten Metriken sind. In diesen Gebäudeblöcken befinden sich dann Subpakete und Klassen, welche wiederum als kleinere Gebäudeblöcke oder einzelne Gebäude dargestellt werden. *ExplorViz* [Hasselbring u. a. 2020] ist ein Softwarevisualisierungswerkzeug, dass diesen Ansatz nutzt. Hierbei verwendet ExplorViz dynamische Analysetechniken, um mit Hilfe von Echtzeitdaten die Kommunikation in Softwarelandschaften zu visualisieren. Eine Kollaborationsfunktion ermöglicht es außerdem, dass gleichzeitig mehrere Entwickler eine Visualisierung betrachten können [Hasselbring u. a. 2020].
Fällt einem Entwickler nun etwas an der Software auf, hat er die Möglichkeit dies analog mit Stift und Papier zu notieren oder es durch ein Issue in einem angebundenen Git-Repository zu vermerken. Um gut dargestellte Fehlerquellen mit dem Entwicklerteam zu teilen, gibt es bisher nur die Möglichkeit einen Screenshot zu erstellen, welcher dem Issue angehängt werden kann. Beide Fälle resultieren darin, dass ein anderer Entwickler das Notierte oder das im Screenshot Dargestellte zuerst nachstellen muss, um aus dieser Ansicht weiter damit zu arbeiten.

In dieser Bachelorarbeit befassen wir uns mit der Erweiterung von ExplorViz durch eine Annotationsfunktion. Durch diese Funktion ist es möglich in der *Landscape*, welche die Visualisierung der Architektur einer betrachteten Software ist, Annotationen zu erstellen, die entweder ungebunden in der Umgebung oder zugehörig zu einem Objekt der Landscape sind. Außerdem ermöglichen Annotationen den direkten Austausch innerhalb einer kollaborativen Sitzung zwischen mehreren Benutzern. Des Weiteren befassen wir uns mit der Persistierung von Snapshots und benutzerdefinierten Token für Git-Hosting-

Anbindungen. Ein *Snapshot* ist hierbei eine Momentaufnahme der Landscape, durch die das Teilen von Darstellungen in ExplorViz optimiert wird. Die Snapshotfunktion wird in der Bachelorarbeit von Julius Brehme [Brehme 2024] behandelt, welche zusammen mit dieser Bachelorarbeit eine thematische Grundlage in einem Bachelorprojekt bildete.
Abschließend evaluieren wir die hinzugefügten Funktionen mittels einer Usability-Studie. In dieser werden Probanden Aufgaben und Fragen zu den implementierten Funktionen gestellt, wodurch wir wertvolle Informationen über die Anwenderfreundlichkeit und Nutzbarkeit dieser bekommen.

1.2. Dokumentenstruktur

Zunächst definieren wir in Kapitel 2 die angestrebten Ziele. In Kapitel 3 werden anschließend die grundlegenden Konzepte und Technologien, welche für ExplorViz verwendet und für die Ziele benötigt werden, erläutert. Im 4. Kapitel präsentieren wir zunächst unseren Ansatz der Annotationsfunktion und erläutern anschließend dessen Implementierung. Kapitel 5 folgt dem gleichen Aufbau für die Persistierung der Snapshots und der benutzerdefinierten Git-Hosting-Token. Anschließend werden in Kapitel 6 die Methode und die Durchführung der Evaluation erläutert und die Ergebnisse präsentiert. Das Kapitel 7 handelt von verwandten Arbeiten. Abschließend ziehen wir in Kapitel 8 ein Fazit und geben einen Ausblick auf mögliche, zukünftige Arbeiten.

Kapitel 2

Ziele

2.1. Z1: Annotationen in der Landscape

In einer Landscape sollen Annotationen als lose Fenster hinzugefügt werden können. Es soll möglich sein, eine Annotation zugehörig zu einem Objekt der Landscape zu erstellen. Außerdem soll eine Annotation auch ohne ein zugehöriges Objekt erstellt werden können. Die Annotationen sollen übersichtlich sein und eine benutzerfreundliche Bedienbarkeit haben.
In einer kollaborativen Sitzung soll es möglich sein, Annotationen zu erstellen und mit anderen Teilnehmern zu teilen. Die Teilnehmer können zudem die geteilten Annotationen für alle anderen Teilnehmer bearbeiten und aktualisieren sowie löschen.

2.2. Z2: Persistierung der Snapshots

Benutzern von ExplorViz soll es ermöglicht werden, eine Landscape mit ihren Einstellungen, geöffneten Fenstern, und Ansichten als Snapshot zu persistieren. Ziel dieser Bachelorarbeit ist die Implementierung einer funktionierenden Persistierung der Snapshots im Backend. Zur Erreichung des Zieles müssen zunächst die benötigten Daten ermittelt werden, um daraus die notwendige Datenbank zu erstellen. Neben der Erstellung einer Datenbank ist ein weiterer Schritt zur Erreichung des Ziels, eine Schnittstelle zum Frontend zu implementieren.

2.3. Z3: Persistierung von benutzerdefinierten Git-Hosting-Token

Die Benutzer von ExplorViz sollen die Möglichkeit bekommen, für verschiedene Projekte aus Git-Hosting-Instanzen, einen API-Token zu hinterlegen. Hierfür ist ein Ziel dieser Arbeit die Implementierung einer funktionierenden Persistierung benutzerdefinierter Git-Hosting-Token. Wie in Ziel 2 müssen zunächst die benötigten Daten ermittelt werden und anschließend aus diesen die notwendige Datenbank erstellt werden. Eine Schnittstelle zum Frontend für die Kommunikation der Daten muss ebenfalls implementiert werden.

2.4. Z4: Evaluation

Nach Erfüllung der vorangegangenen Ziele versucht die Evaluation zu prüfen, ob die Erweiterung von ExplorViz durch die implementierten Funktionen den angestrebten Nutzen erfüllt. Hierbei ist der angestrebte Nutzen für die Annotationsfunktion eine benutzerfreundliche Funktion für Annotationen zu haben, die in kollaborativen Sitzungen für den Informationsaustausch oder -erhalt genutzt werden kann. Für die Snapshotfunktion ist der angestrebte Nutzen ebenfalls, dass diese Funktion benutzerfreundlich bedient werden kann und dass das Teilen von Landscapes vereinfacht wird. Gleiches gilt für die Erweiterung der Git-Hosting-Anbindung und ihrer benutzerdefinierten Token. Die Evaluation zielt somit insbesondere auf die Benutzerfreundlichkeit und Nutzbarkeit ab.

Kapitel 3

Grundlagen und Technologien

3.1. Persistierung

Im Umgang mit Softwaresystemen werden viele Daten verschiedener Typen benötigt. Diese Daten sind für das Programm zur Ausführungszeit abrufbar und oft auch nur in diesem Zeitraum von Relevanz. Jedoch gibt es Anwendungsfälle, bei denen bestimmte Daten auch nach der Ausführung eines Programms erhalten bleiben sollen. Hierfür werden die Daten meist in Dateien gesichert, sodass diese bei späterer Programmnutzung wieder abgerufen werden können. Man spricht in diesem Fall von Persistierung. Dabei untersteht die Persistierung gewissen Anforderungen: (1) Beliebige Typen können persistiert werden und nicht nur bestimmte; (2) Jeder Typ wird bei der Persistierung gleich behandelt; (3) Wenn ein Wert persistiert wird, dann wird auch sein beschreibender Typ persistiert. Letzteres ist darin begründet, dass es nicht möglich sein soll einen Wert mit einem bestimmten Typen zu persistieren und mit einem anderen wieder abzurufen [Atkinson und Buneman 1987].

3.2. Programmierschnittstelle

Eine Grundlage für die Kommunikation zwischen Softwaresystemen bieten Programmierschnittstellen, im Folgenden API (abgeleitet von dem englischen Application Programming Interface) genannt, die in moderner Software intensiv genutzt werden [Myers und Stylos 2016]. Eine API ist eine Schnittstelle einer Anwendung oder einer Bibliothek von Funktionen, die Entwicklern Zugriff auf ihre Dienste und Daten gewährt [Stylos u. a. 2009; Robillard 2009]. Hierbei stellt die API vordefinierte Ressourcen wie Methoden, Objekte, oder URIs bereit. Die Nutzung dieser Ressourcen ermöglicht es, dass andere Applikationen auf die Daten und Dienste zugreifen können, ohne dass sie diese selbst implementieren müssen [Meng u. a. 2018]. Außerdem liefern APIs ein hohes Abstraktionslevel zur Vereinfachung von Programmieraufgaben und tragen zur Vereinheitlichung von Programmiererfahrungen bei, indem für bestimmte Aufgaben eine einheitliche Schnittstelle vorhanden ist [Robillard 2009]. Hierdurch sind APIs zentral für viele moderne Softwarearchitekturen und unterstützen neben der Wiederverwendung von Code auch das Design von verteilten und modularen Softwareanwendungen [Meng u. a. 2018].

3.3. Microservice

Microservices sind ein Stil in der Softwarearchitektur, welcher ein wichtiges Paradigma für die Realisierung von Softwareanwendungen geworden ist [Hilbrich und Lehmann 2022]. Es gibt viele verschiedene Definitionen von Microservices, weshalb im Folgenden erläutert wird, wie wir im Zuge dieser Bachelorarbeit Microservices verstehen. Die Grundlage von Microservices besteht darin, ein System vertikal in verschiedene Teile, die jeweiligen Microservices, aufzuteilen. Hierbei wird das System so aufgeteilt, dass ein einzelner Microservice für eine bestimmte Aufgabe oder einen bestimmten Aufgabenbereich zuständig ist. Dies kann von der Darstellung einer Anwendung bis hin zur Persistierung der Daten reichen [Hilbrich und Lehmann 2022]. Der einzelne Microservice ist unabhängig von anderen Diensten bereitstellbar [Lewis und Fowler 2014] und kann über einen oder mehrere Netzwerkendpunkte mit anderen Diensten nachrichtenbasiert kommunizieren [Nadareishvili u. a. 2016]. Aus dieser Definition heraus ermöglicht die Microservice-Architektur eine bessere Skalierbarkeit der einzelnen Microservices und die Möglichkeit bestimmte Aufgabenbereiche einer Softwareanwendung einzelnen Entwicklerteams zuzuordnen. Damit geht eine Verringerung der Abhängigkeiten zwischen den einzelnen Teams und somit eine Steigerung der Effizienz einher. Diese Zuordnung ermöglicht es außerdem, dass Entwickler ihre jeweils bevorzugten oder die für den Anwendungsfall optimalen Programmiersprachen benutzen können [Nadareishvili u. a. 2016].

3.4. Quarkus

Quarkus[1] ist ein Open-Source Java Framework, das sich offiziell selbst als einen Kubernetes-nativen Java Stack, der auf OpenJDK HotSpot und GraalVM zugeschnitten ist, beschreibt. Es wurde speziell für Cloud-natives und Serverless Computing konzipiert und ist als Open-Source Projekt offen erweiterbar. Hierbei wird der Fokus besonders auf schnelle Startzeiten und einen geringen Ressourcenverbrauch gelegt. Dies wird dadurch erreicht, dass Quarkus im Build-Prozess Arbeitsschritte macht, die in traditionellen Frameworks erst zur Laufzeit passieren. Dies umfasst zum Beispiel das Scannen von Klassenpfaden, das Parsing von Konfigurationen, oder auch das Ersetzen von Reflection-Aufrufen durch reguläre Aufrufe. Da Quarkus auf einen modernen, lebensnahen Ansatz für die Java-Entwicklung abzielt, ermöglicht es die effiziente Entwicklung von Microservices und serverlosen Anwendungen.

3.5. MongoDB

MongoDB[2] ist eine Open-Source dokumentenorientierte Datenbank und wird als NoSQL-Datenbank kategorisiert. Die Daten in einer MongoDB werden intern im binären BSON-

[1] https://quarkus.io/
[2] https://www.mongodb.com/

Format, welches verwandt zu JSON ist, verwaltet. Da BSON mehr Datentypen als JSON abdeckt, ermöglicht MongoDB flexibel verschiedene Datentypen miteinander zu kombinieren und zu speichern. Außerdem ermöglicht das Format eine hohe Skalierbarkeit. Die Datenbank besteht aus Collections, die analog zu Tabellen aus relationalen Datenbanken sind. Eine Collection besteht wiederum aus mehreren Dokumenten, die den Tabelleneinträgen einer relationalen Datenbank ähnlich sind, aber als Schlüssel-Wert-Paare gespeichert werden. Da die Dokumente schemalos sind, können mehrere Dokumente einer Collection verschieden aufgebaut sein. Außerdem bietet MongoDB die Möglichkeit, verschachtelte Daten zu speichern, wodurch das Arbeiten mit den Daten effizienter ist als in relationalen Datenbanken wie SQL. MongoDB kann somit strukturierte, halbstrukturierte, und unstrukturierte Daten verarbeiten.

3.6. NestJS

NestJS[3] ist ein Open-Source JavaScript Framework zum Entwickeln von effizienten und skalierbaren *Node.js*[4] Serveranwendungen. Dabei kombiniert NestJS Elemente der objektorientierten, der funktionalen, und der funktionalen reaktiven Programmierung. Da es auf verschiedenen Node.js Frameworks aufbaut, ermöglicht es einen freien Umgang mit anderen Modulen von Drittanbietern. Die Node.js Frameworks sind für Entwickler direkt zugänglich. NestJS Anwendungen bestehen unter anderem aus Controllern, Providern, und Modulen. Die Controller sind für die Bearbeitung einkommender Nachrichten und deren Antworten zuständig. Provider hingegen sind für die Datenbeschaffung zuständig. Die Dependency Injection von NestJS ermöglicht es hierbei, dass durch die Provider verschiedene Beziehungen zwischen Objekten erstellt werden können. Ein Modul stellt eine Klasse dar, die mit der *@Module*-Markierung annotiert wird. Hierdurch werden Metadaten geliefert, damit NestJS die Klasse in die Anwendungstruktur organisieren kann.

3.7. Redis

Redis[5] ist eine performante In-Memory-Datenbank, die zu den NoSQL-Datenbanken gehört. Es bietet komplexe Datenstrukturen wie Strings, Hashes, Listen, Bitmaps, und viele mehr an. Auf diesen Strukturen besteht außerdem die Möglichtkeit atomare Operationen, wie das Inkrementieren eines Hash-Werts, auszuführen. Redis bietet für die Persistierung von Daten das periodische Speichern oder das Speichern in einer Append-Only-File (AOF), die einem Eventlog gleichkommt, an. Zudem bietet es die Möglichkeit die Persistierung auszuschalten, falls lediglich ein In-Memory-Cache gebraucht wird. Es kommt mit Built-In Funktionen wie Transaktionen, Lua Skripting, oder auch Replication. Letzteres ermöglicht es, durch

[3] https://nestjs.com/
[4] https://nodejs.org/
[5] https://redis.io/

asynchrone Replikation, Redis in verteilten Systemen zu benutzen. Außerdem liefert Redis die Funktion Redis Pub/Sub, welche das Publish/Subscribe-Messaging Paradigma implementiert. Das Publish/Subscribe-Messaging Paradigma ist ein Mechanismus für die asynchrone Kommunikation zwischen Diensten, bei dem der Sender und der Empfänger voneinander separiert werden und ein Broker die Nachrichten zwischen diesen verteilt [Kul und Sayar 2021].

3.8. Ember.js

Ember.js[6] ist ein Open-Source JavaScript Framework für die Erstellung von Webanwendungen. Es basiert auf dem MVVM-Muster (Model-View-ViewModel), welches das MVC-Muster (Model-View-Controller) erweitert. In diesem werden die Controller- und View-Einheiten zusammengefasst und die gesamte Logik der Darstellung aus dem Controller in ein ViewModel-Objekt ausgelagert. Zur Kommunikation zwischen dem Modell und dem View/Controller wird das ViewModel-Objekt genutzt [Mishra 2017]. Ember.js bietet mit der Ember CLI die Möglichkeit, neue Code-Entitäten zu generieren und die dafür nötigen Dateien an den richtigen Orten zu erstellen. Hierbei werden für die umfangreiche Testumgebung, die das Framework bietet, automatisch Tests zu den jeweiligen Entitäten erstellt. Der Ember Router liefert das asynchrone Laden von Daten mittels dynamischen URL-Segmenten und Queryparametern. Zudem werden verschachtelte URLs mit inkrementellen Datenaufrufen unterstützt. Ember.js bietet außerdem mit Ember Data eine Bibliothek an, um auf Datenebene den Datenzugriff auf mehrere Quellen gleichzeitig zu ermöglichen. Dadurch kann sichergestellt werden, dass die Datenmodelle einer Anwendung an jeder Stelle auf dem aktuellsten Stand sind. Die Performanz des Frameworks wird durch die Nutzung der *Glimmer*[7] Rendering Engine sichergestellt.

3.9. three.js

three.js[8] ist eine JavaScript Bibliothek zur Darstellung von 3D-Inhalten in Webanwendungen. Sie nutzt die JavaScript Programmierschnittstelle *WebGL*[9], die für die Darstellung von 3D-Grafiken im Browser zuständig ist, um dreidimensional zu zeichnen. Da WebGL nur Punkte, Linien, und Dreiecke zeichnen kann, erweitert three.js diese. Hierbei ermöglicht three.js die Nutzung von Szenen, Licht, Schatten, verschiedenen Materialien, und Texturen sowie Weiterem, ohne dass diese selbst neu implementiert werden müssen. Eine einfache three.js Anwendung besteht aus einem Renderer, dem ein Kamera-Objekt und eine Szene übergeben wird, wie in Abbildung 3.1 zu sehen. Hierdurch wird der durch die Kamera

[6] https://emberjs.com/
[7] https://glimmerjs.com/
[8] https://threejs.org/
[9] https://khronos.org/webgl/

sichtbare Inhalt der 3D-Szene in einen HTML-Canvas vom Renderer gezeichnet. Die Szene kann durch weitere Objekte wie Licht und Mesh erweitert werden, wobei ein Mesh eine bestimmte Geometrie mit einem Material und gegebenenfalls einer Textur darstellt.

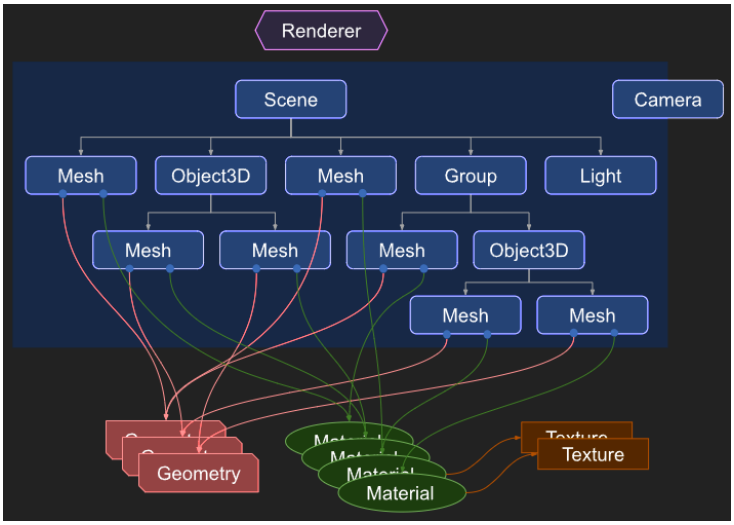

Abbildung 3.1. Aufbau der Objekte einer einfachen three.js Anwendung [three.js 2024].

3.10. ExplorViz

ExplorViz [Hasselbring u. a. 2020] ist ein laufendes Forschungsprojekt der Software Engineering Gruppe an der Christian-Albrechts-Universität zu Kiel (CAU Kiel). Das Ziel ist es Entwicklern und Anwendern beim System- und Programmverständnis von Softwarelandschaften oder einzelnen Anwendungen zu helfen. Hierbei bietet es zudem Details über die Kommunikationen innerhalb der Anwendung [Hasselbring u. a. 2020]. Um dies zu ermöglichen, stellt ExplorViz die Softwarelandschaft als 3D-Modell im Sinne der City Metapher dar [Wettel und Lanza 2007]. Hierfür werden zur Laufzeit die strukturellen und dynamischen Daten der Anwendung genutzt [Krause-Glau und Hasselbring 2022]. Im Folgenden geben wir einen kurzen Überblick über die Architektur von ExplorViz und über den Ansatz der Visualisierung.

3.10.1. Architektur

Die Entwicklung von ExplorViz startete 2012 zunächst als monolithische Software, die auf Java und GWT[10] basierte. In 2018 wurde die Software dann in eine Microservice-Architektur

[10]https://www.gwtproject.org/

umstrukturiert. Dies passierte unter anderem durch das Wegfallen des JavaScript Native Interface in GWT3, welches zur Nutzung von JavaScript Bibliotheken in ExplorViz verwendet wurde [Zirkelbach u. a. 2018]. Neben dem Frontend, welches auf dem JavaScript Framework Ember.js basiert, bilden verschiedene Microservices das Backend. Abbildung 3.2 zeigt die vorhandenen Microservices und den Datenfluss für die Kommunikation zwischen diesen. Das Java Framework Quarkus bildet für die meisten Services des Backends die Basis, so zum Beispiel für den User Service, den Span Service, oder auch den Code Service. Der Collaboration-Service basiert hingegen auf dem JavaScript-Framework NestJS.

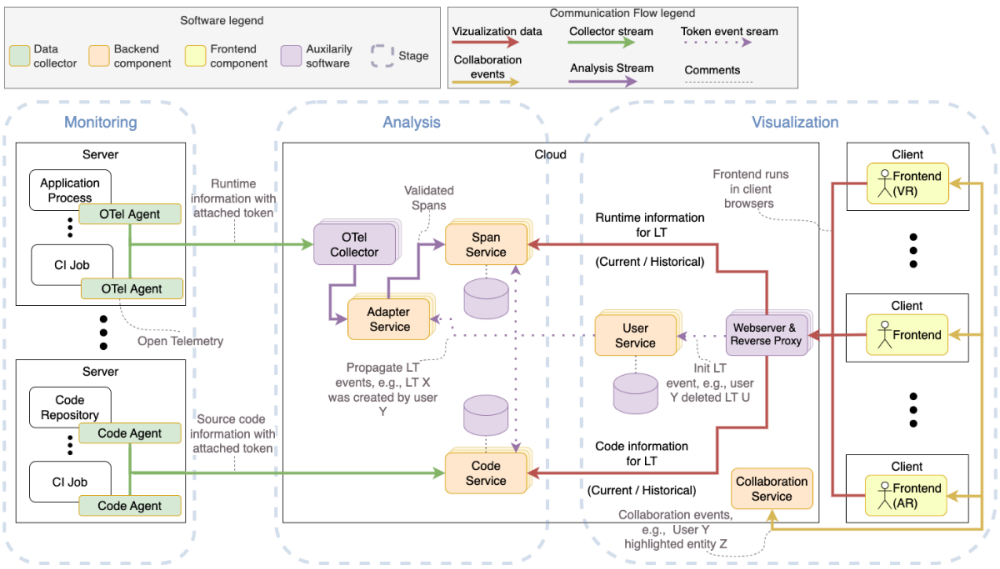

Abbildung 3.2. Konzeptuelles Design des Softwarevisualisierung-as-a-Service Ansatzes ExplorViz [ExplorViz 2024].

Laufzeitdaten werden in ExplorViz nach dem OpenCensus[11] Standard gesammelt und an den Adapter Service weitergereicht. Dieser validiert einen Landscape Token, welcher beim User Service angefragt wird, den der Benutzer für diese bestimmte Landscape erhalten hat. Wenn der Landscape Token und die gesammelten Daten valide sind, werden diese in Strukturdaten und dynamische Daten aufgeteilt und an den Span Service weitergereicht. Die Kommunikation zwischen den Services wird über Apache Kafka[12] Streams realisiert. Der Span Service verwaltet und persistiert die Strukturdaten und dynamischen Daten. Zuletzt erhält das Frontend vom Span Service die Daten, um diese mithilfe von three.js darzustellen.
Für die Kollaborationsfunktion von ExplorViz gibt es den Collaboration-Service. Die-

[11] https://opencensus.io/
[12] https://kafka.apache.org/

ser verwaltet die Räume und die zugehörigen Benutzeranfragen und kommuniziert als Websocket-Server mit dem Frontend.

3.10.2. Visualisierung

Abbildung 3.3. Beispielhafte Visualisierung aus der Demo von ExplorViz in der Browseransicht.

Die Visualisierung von ExplorViz wird mittels WebGL in einem HTML-Canvas gezeichnet, wobei die Objekte, die für die Darstellung der visualisierten Anwendungen benötigt werden, mit three.js realisiert werden. Ursprünglich bestand die Visualisierung aus zwei verschiedenen Ansichten, der Landscape Ansicht, die eine Übersicht der Kommunikation zwischen den Anwendungen gab und der Application Ansicht, die den Aufbau einer Anwendung zeigt [Fittkau u. a. 2015]. Dies wurde restrukturiert, sodass es nur noch eine Ansicht gibt, bei der die Kommunikation zwischen den Anwendungen visuell mittels Linien dargestellt wird. ExplorViz verfügt über drei Darstellungsmodi, der Darstellung im Browser, dem Virtual Reality (VR) Modus, und dem Augmented Reality (AR) Modus. Somit ist es möglich, ExplorViz in einem Browser auf dem Bildschirm, über einen WebXR[13]fähigen Browser mit einer VR-Brille, oder zum Beispiel mittels eines Smartphones den AR Modus zu benutzen [Krause-Glau u. a. 2022].

Wie in Abbildung 3.3 zu sehen, besteht die visuelle Darstellung einer Landscape aus ein oder mehrerer grauer Fundamentboxen (1), die eine Anwendung darstellen. Die Anwendungen enthalten jeweils Pakete und Subpakete (2), hier in grün und blau dargestellt, die wiederum Klassen (3) enthalten, deren Größe abhängig ihrer instanziierten Objekte

[13]https://immersiveweb.dev/

entspricht. Pakete können ihre Inhalte verstecken (linke 2) oder anzeigen (rechte 2). Die Kommunikation zwischen Klassen wird mittels bidirektionalen Linien (4) dargestellt. ExplorViz bietet außerdem die Möglichkeit Pop-ups (5) zu allen bisher genannten Objekten zu öffnen. Während die visuelle Darstellung der Objekte einen Einblick in bestimmte Werte geben soll, wie zum Beispiel die Höhe der Klassen, können die genauen Werte den Pop-ups entnommen werden. Über den Einstellungsbutton (7) hat der Benutzer die Möglichkeit, einen Kollaborationsraum zu öffnen oder bestehenden beizutreten, sich mit der IDE Visual Studio Code zu verbinden, den Restructuring-Modus zu aktivieren, oder die benutzerdefinierten Einstellungen zu öffnen. Befindet sich der Benutzer in einem Kollaborationsraum, so verfügen die Pop-ups über einen Teilen-Button (6), um diese mit anderen Teilnehmern zu teilen. Mit dem linken Button (8) kann der Benutzer Filter einstellen, nach bestimmten Anwendungen suchen, oder sich die Zeitstempel der Entitäten einer Anwendung anzeigen lassen. Die Timeline (9) visualisiert die Anzahl der Anfragen im zeitlichen Ablauf. Über die Zeitstempel im Graphen lässt sich ein bestimmter Zeitpunkt grafisch darstellen.

Außerdem zeigt Abbildung 3.4 die Möglichkeit einer Heatmap (1) Funktion, durch die abhängig der Metriken, die auch im Pop-up sichtbar sind, die Darstellung der Objekte eingefärbt wird. Weiterhin bietet ExplorViz ein Kontextmenü (2), über welches zum Beispiel alle Komponenten sichtbar gemacht werden können.

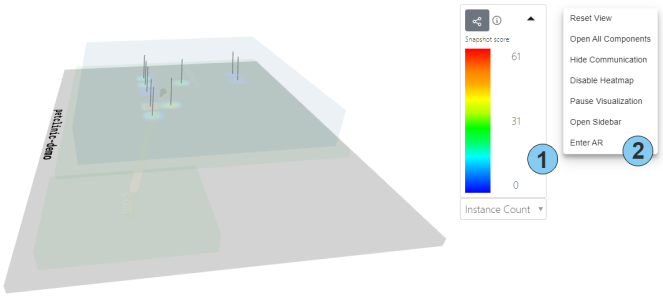

Abbildung 3.4. Beispielhafte Visualisierung der Heatmap und des Kontextmenüs in der Browseransicht.

Kapitel 4

Annotationen

Ein Ziel dieser Arbeit war die Entwicklung einer kollaborativen Annotationsfunktion. Im Folgenden erläutern wir den Ansatz, mit dem dieses Ziel erreicht wurde und gehen auf die Implementierung dieses Ansatzes ein.

4.1. Ansatz

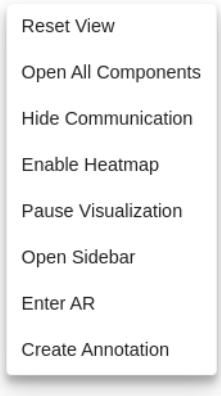

Abbildung 4.1. Ansicht des Kontextmenüs in der Landscape von ExplorViz.

Die Annotationsfunktion haben wir in Form von Fenstern, die in der Landscape per Drag&Drop bewegt werden können, umgesetzt. Für diesen Ansatz konnten wir uns an den bereits vorhandenen Pop-ups orientieren, da diese von der Struktur und den Funktionen sehr ähnlich sind. Es gibt zwei Arten von Annotationen, die freien Annotationen und die zugehörigen Annotationen. Freie Annotationen können, wie in Abbildung 4.1 zu sehen ist, über das Kontextmenü erstellt werden. Sie gehören zu keinem Objekt der Landscape und sind in dieser frei bewegbar. Die zugehörigen Annotationen können, wie auch die Pop-us, durch das Hovern über Objekte der Landscape erstellt werden. Hierfür muss nach dem Hovern, wie in Abbildung 4.2 zu sehen, die Shift-Taste gedrückt werden, damit beim Bewegen der Maus das Fenster nicht gelöscht wird. Wurde das Fenster auf diese Weise das erste Mal bewegt, bleibt es bestehen, bis es manuell über den Löschen-Button

entfernt wird. Für die zugehörigen Annotationen hatten wir ursprünglich die Idee, diese ebenfalls über das Rechtsklickmenü erstellen zu lassen, indem man einen Rechtsklick auf das jeweilige Objekt in der Landscape macht. Dies haben wir allerdings nicht so umgesetzt, da es zu Problemen mit der aktuellen Implementierung des Rechtsklickmenüs kam. In dieser wurden zwar beim Hovern über ein Objekt die Informationen zu dem Objekt abgerufen, allerdings konnten diese Informationen nicht bei einem Rechtsklick mit abgerufen werden. Deshalb wurde sich an dem vorhandenen Konzept für die Erstellung der Pop-ups orientiert.

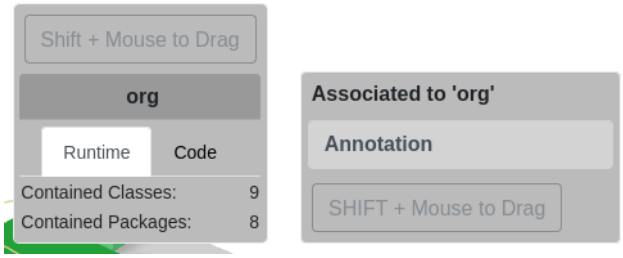

Abbildung 4.2. Darstellung eines ungeöffneten Pop-ups (links) und einer ungeöffneten Annotation (rechts) beim Hovern über ein Objekt in der Landscape.

Die zugehörigen Annotationen gehören zu einem Objekt der Landscape. Damit der Benutzer die Übersicht behält, welche Annotation zu welchem Objekt gehört, war ursprünglich angedacht diese mit Linien zu verbinden. Hiergegen haben wir uns während der Entwicklung entschieden, da schon bei wenigen Annotationen die Übersichtlichkeit durch die Annotationslinien zusammen mit den Kommunikationslinien eingeschränkt werden würde. Auch durch die freie Beweglichkeit der Annotationen und der dadurch verbundenen Bewegung der Linien würde diese Übersichtlichkeit weiter eingeschränkt werden. Eine weitere Idee war das Einfärben des betroffenen Objekts. Da dies allerdings mit der Highlight-Funktion von ExplorViz kollidieren kann, haben wir uns ebenfalls dagegen entschieden. Anstelle dessen haben wir uns als Orientierung der zugehörigen Annotationen dafür entschieden, dass es möglich ist, das Objekt mit der vorhandenen Ping-Funktion hervorzuheben. Außerdem kann durch das Hovern über eine zugehörige Annotation das jeweilige Objekt ebenfalls hervorgehoben werden. Dies funktioniert auch andersherum durch das Hovern über ein Objekt. Als letzte Orientierungsmöglichkeit wird bei der Erstellung einer zugehörigen Annotation das Label des jeweiligen Objekts um einen Hinweis erweitert.

4.1.1. Annotationsfenster

Wie zuvor erwähnt haben wir die Annotationsfunktion mittels Fenstern realisiert. Hierbei war das Ziel, einen übersichtlichen Aufbau der Fenster für die jeweilige Annotationsart zu erstellen, der die gesamte Funktionalität offen bereitstellt. Wie in Abbildung 4.3 zu sehen,

besteht eine Annotation immer aus einem Titel und einem Text. Beide Eingabefelder sind nicht auf eine Länge beschränkt und werden im Ansichtsmodus scrollbar, sofern die Länge die mögliche Anzeigegröße übersteigt. Außerdem liefert ein Annotationsfenster immer die Information über den Ersteller und die Person, die zuletzt die Annotation bearbeitet hat. Hierauf werden wir in Abschnitt 4.1.3 genauer eingehen. Über einen Button am unteren Fensterrand ist es möglich, zwischen dem Editiermodus und dem Ansichtsmodus zu wechseln. Eine neu erstellte Annotation befindet sich zunächst immer im Editiermodus. Jede Annotation verfügt über jeweils einen Button zum Teilen, zum Minimieren, und zum Entfernen. Eine zugehörige Annotation stellt zudem ein Label, das angibt zu welchem Objekt die Annotation gehört und einen Button für die Ping-Funktion bereit.

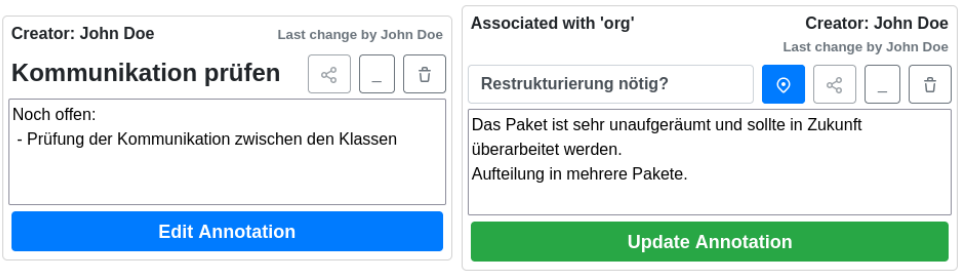

Abbildung 4.3. Darstellung einer freien Annotation im Ansichtsmodus (links) und einer zugehörigen Annotation im Editiermodus (rechts).

4.1.2. Minimierungsfunktion

Da Softwareanwendungen sehr komplex sein und viele Komponenten haben können, kann es schnell unübersichtlich werden, wenn es zu einigen der Komponenten Annotationen gibt. Um diese Unübersichtlichkeit zu minimieren, haben wir die Möglichkeit geschaffen, dass Annotationsfenster minimiert werden können. Aufgrund der verschiedenen Eigenschaften der beiden Annotationsarten unterscheidet sich die Art und Weise wie eine Annotation minimiert wird.

Abbildung 4.4. Darstellung einer freien Annotation in minimierter Ansicht.

Da freie Annotationen frei in der Landscape existieren und nicht zu Objekten zugeordnet sind, haben wir die Minimierungsfunktion für diese so umgesetzt, dass der Annotationstext versteckt wird. Das Fenster klappt somit die unteren zwei Drittel der Annotation ein, wie

in Abbildung 4.4 zu sehen ist, ist aber weiterhin frei in der Landscape bewegbar.
Anders ist dies bei den zugehörigen Annotationen. Da diese fest zu einem Objekt gehören, ist es möglich, diese aus der Ansicht zu entfernen. Damit der Benutzer trotzdem die Übersicht behalten kann, zu welchem Objekt eine Annotation existiert, wird bei Erstellung einer Annotation der Labeltext des Objekts mit dem Hinweis *'[annotated]'* erweitert. Dies zeigt Abbildung 4.5. Ein erneutes Öffnen einer minimierten Annotation funktioniert genauso wie das Erstellen einer zugehörigen Annotation.

Abbildung 4.5. Darstellung eines Pakets in der Landscape, zu dem eine Annotation erstellt wurde.

4.1.3. Kollaborationsfunktion

Da ExplorViz die Möglichkeit der Kollaboration bietet, sollten auch die Annotationen in diesen Kontext eingearbeitet werden. Hierfür gibt es, wie bei den Pop-ups, einen Teilen-Button. Anders als bei den Pop-ups, bei denen dieser erst sichtbar ist, sobald ein Raum offen und das Pop-up angepinnt ist, ist dieser bei einer Annotation immer sichtbar. Sollte sich die aktuelle Landscape nicht in einem Kollaborationsraum befinden oder die Annotation bereits geteilt sein, wird der Button ausgegraut, ansonsten ist dieser blau hinterlegt. In diesem Fall kann der Benutzer über den Button das Fenster den anderen Teilnehmern im Raum bereitstellen. Hierbei wird die Annotation in der Landscape der anderen Teilnehmer immer im Ansichtsmodus erstellt.

Damit es zu keinen Synchronisierungsproblemen kommt, kann immer nur ein Benutzer zur Zeit im Editiermodus einer Annotation sein. Sollte ein anderer Benutzer diesen öffnen wollen, während bereits jemand in diesem ist, erscheint am unteren rechten Bildschirmrand eine Fehlermeldung, die in Abbildung 4.6 gezeigt ist. Beendet der Benutzer den Editiermodus, indem er in den Ansichtsmodus der Annotation wechselt, wird die Anzeige über den letzten Bearbeiter auf dessen Anzeigenamen gesetzt und die Änderungen mit den anderen Teilnehmern geteilt.

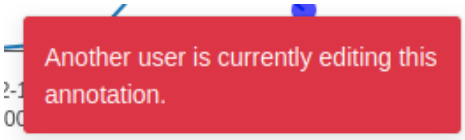

Abbildung 4.6. Fehlermeldung bei dem Versuch den Editiermodus einer Annotation zu öffnen, wenn bereits ein anderer Benutzer des Kollaborationsraumes in dem Editiermodus der Annotation ist.

Um zu verhindern, dass eine Annotation im Editiermodus vergessen wird, sodass diese für alle anderen Teilnehmer unveränderbar wäre, ist es nicht möglich die Annotation im Editiermodus zu minimieren. Hierbei wird die Fehlermeldung aus Abbildung 4.7 unten rechts angezeigt. Ansonsten steht es den Teilnehmern eines Raumes frei, die Annotationen zu minimieren, ohne dass hierbei die Ansicht der anderen Teilnehmer beeinflusst wird.

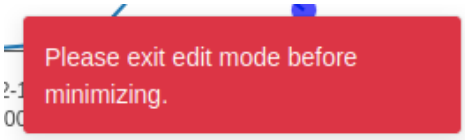

Abbildung 4.7. Fehlermeldung beim Versuch eine Annotation zu minimieren, während der Editiermodus geöffnet ist.

4.2. Implementierung

Für die Implementierung der Annotationsfunktion haben wir das Frontend von ExplorViz erweitert. Hierfür wurden die Ember.js Komponente *AnnotationCoordinatorComponent*, der Ember.js Service *AnnotationHandlerService*, und die Klasse *AnnotationData* erstellt. Eine Ember.js Komponente besteht dabei aus zwei Teilen, einer *.hbs*-Datei, die als Template für die Darstellung dient und einer JavaScript, bzw. in diesem Fall einer TypeScript-Datei, die das Verhalten der Komponente definiert. Der *AnnotationHandlerService*, im Folgenden nur noch *AnnotationHandler* genannt, ist für die Logik hinter einer Annotation zuständig. Da es Probleme beim Laden der Annotationen aus Snapshots gab, haben wir uns für die Implementierung als Service entschieden. So wird der Service einmal gestartet und ist von allen Komponenten der Anwendung nutzbar, sodass immer eine einheitliche Liste aller Annotationen abrufbar ist. Der Service wird beim Aufruf der *initRendering*-Methode, die zum Start einer Landscape aufgerufen wird, initialisiert.

4.2.1. Annotation Data

Damit die Annotationen einen Titel und Text haben und ihre weiteren Funktionen bereitstellen können, bietet die Klasse *AnnotationData* verschiedene Attribute, die eine Annotation beschreiben. Das Klassendiagramm in Abbildung 4.8 zeigt, dass in der Klasse unter anderem festgehalten wird, ob eine Annotation ein zugehöriges Objekt hat. Dies wird mit dem Attribut *isAssociated* und den Attributen für die Typen *EntityMesh* und *Entity* realisiert. Es sei hier erwähnt, dass der Typ *Entity* so nicht existiert, sondern das zugehörige Attribut in der Klasse einen der erbenden Typen annehmen kann. Neben den beiden Attributen *annotationText* und *annotationTitle*, die den Text und den Titel einer Annotation halten, werden die anderen Attribute für die Darstellung und das Verhalten einer Annotation genutzt. Auf die jeweiligen Einsätze wird im Rest dieses Abschnitts eingegangen.

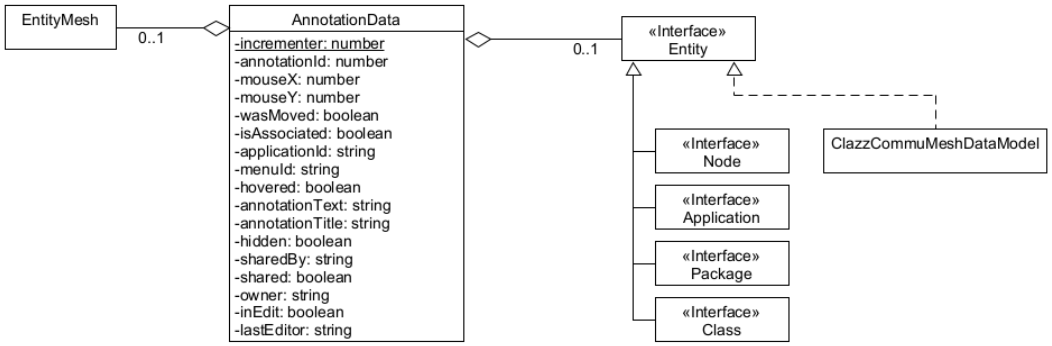

Abbildung 4.8. Klassendiagramm für die Klasse *AnnotationData*.

Die Klasse verfügt außerdem über ein statisches Attribut *incrementer*. Dieses wird genutzt, um bei der Initialisierung das Attribut *annotationId* zu setzen und wird bei Start des Systems standardmäßig mit 0 initialisiert. In Fällen wie dem Laden eines Snapshots, in dem Annotationen gespeichert wurden, ist dieses wichtig. Beim Laden eines Snapshots werden die Annotationen, die zuvor einmal persistiert wurden, erneut erstellt. In diesem Fall bekommt der Konstruktor der Klasse eine ID übergeben, welche für das Attribut gesetzt wird. Der gleiche Wert wird inkrementiert und anschließend als Wert für das Attribut *incrementer* gesetzt. Dies hat den Vorteil, dass nach einem Systemneustart Annotationen aus Snapshots mit den gleichen IDs erstellt werden.

4.2.2. Annotation Handler Service

Für die Verwaltung und Interaktionen mit den Annotationen ist der *AnnotationHandler* zuständig. Über diesen werden neue Annotationen erstellt und bereits vorhandene verändert, minimiert, oder gelöscht. Außerdem liefert dieser die Funktionen, um Annotationen zu

teilen und auf geteilte zu reagieren, wie in Abschnitt 4.2.5 genauer erläutert wird.
Der *AnnotationHandler* besitzt die Liste *annotationData* von *AnnotationData*-Objekten. Diese beinhaltet alle vorhandenen Annotationen bzw. deren Daten und wird für die Darstellung der Annotationen und den Interaktionen mit diesen benutzt. Daneben besitzt der Service noch eine weitere Liste *minimizedAnnotations*, auf die genauer im Abschnitt 4.2.4 eingegangen wird.

Die in Listing 4.1 gezeigte Methode *addAnnotation* wird vewendet, um eine neue Annotation den bestehenden hinzuzufügen. Unter den Parametern, die die Methode beim Aufruf übergeben bekommt, gibt es einige, wie die *annotationId* oder das *mesh*, welche *undefined* sein können. Da es verschiedene Situationen gibt, in denen Annotationen erstellt werden, zum Beispiel durch das Laden eines Snapshots, das Betreten eines Kollaborationsraumes, oder das Erstellen einer Annotation in der Landscape, ist die Möglichkeit, Parameter als *undefined* zu übergeben, wichtig. So existiert beim Laden eines Snapshots bereits eine *annotationId*, die es für die Neuerstellung in der Landscape noch nicht gibt. Wird *addAnnotation* aufgerufen, so wird in den Zeilen 6 bis 12 zuerst überprüft, ob die Annotation minimiert bereits existiert. Hierfür wird zunächst geprüft, ob ein *mesh*, also ein *three.js* Objekt der 3D-Landscape-Umgebung, übergeben wurde. Ist dies der Fall, wird vor der eigentlichen Erstellung einer neuen Annotation erst in der Liste *minimizedAnnotations* nach einer Annotation mit eben diesem *mesh*, bzw. der ID dessen gesucht. Wird eine solche Annotation gefunden, dann wird diese der Liste *annotationData* hinzugefügt und die Variable *minimized* auf *true* gesetzt. Hierbei wird die Annotation noch nicht aus der Liste *minimizedAnnotations* entfernt, da das Hinzufügen bereits beim Hovern aufgerufen wird. Sollte der Benutzer nicht mit Drücken der SHIFT-Taste und Ziehen des Fensters die Annotation final öffnen, so würde die minimierte Version gelöscht werden. Aus diesem Grund wird bei späterer Mausbewegung überprüft, ob eine Annotation sowohl in *annotationData*, als auch in *minimizedAnnotations* vorhanden ist. Wenn dem so ist und *wasMoved* ist wahr für die Annotation, dann wird die Annotation in *minimizedAnnotations* entfernt. Da dies bei jeder Mausbewegung überprüft wird, passiert dies sofort nach dem erneuten Öffnen. Anschließend wird überprüft, ob die zu erstellende Annotation bereits in *annotationData* vorhanden ist. Dies ist immer der Fall, wenn diese vorher in *minimizedAnnotations* gefunden wurde. Es kann aber auch bereits eine geöffnete Annotation zu dem Objekt geben. In diesem Fall wird die Variable *alreadyExists* auf *true* gesetzt. Gibt es anschließend keine Annotation mit der ID des Meshs in *minimizedAnnotations* und in *annotationData* oder ist *mesh* undefiniert, so wird eine neue Annotation erstellt. Hierbei wird in letzterem Fall eine freie Annotation, sonst eine zugehörige Annotation mit den jeweils nötigen Attributen erstellt. Die Zeilen 10 und 29 zeigen das Anhängen der bereits gefundenen oder neuen Annotation an die bestehende Liste.

Listing 4.1. Methode zur Erstellung von Annotationen im *AnnotationHandler*. Durch //... werden ausgelassene Zeilen beschrieben.

```
addAnnotation({annotationId,mesh,position,wasMoved,menuId,hovered,annotationTitle,
    annotationText,sharedBy,owner,shared,inEdit,lastEditor,}: { //... })
{
  let minimized = false;
  let alreadyExists = false;

  if (isEntityMesh(mesh)) {
    const annotation = this.minimizedAnnotations.filter(
        (an) => an.entity?.id === mesh.dataModel.id );
    if (annotation.length === 1) {
      this.annotationData = [...this.annotationData, annotation[0]];
      minimized = true;
    }

    const anno =
        this.annotationData.filter((an) => an.entity?.id === mesh.dataModel.id);
    if (anno.length === 1) { alreadyExists = true; }
  }

  if (!minimized && !alreadyExists) {
    //...
    let newAnnotation;
    if (!isEntityMesh(mesh)) {
      newAnnotation = //... Konstruktoraufruf einer freien Annotation
    } else {
      newAnnotation = //... Konstruktoraufruf einer zugehoerigen Annotation
      //...
    }
    //...
    this.annotationData = [...this.annotationData, newAnnotation];
  }
}
```

4.2.3. Annotationsfenster

Die Darstellung und die Basis der Funktionalität des Fensters für die Annotationen werden in der *AnnotationCoordinatorComponent*, im Folgenden *AnnotationCoordinator* genannt, umgesetzt. Hierbei existiert für jede Annotation, die im *AnnotationHandler* erstellt wird, eine Komponente, die über die Klasse *BrowserRendering* geladen wird. Letztere ist die

Komponente, die für die Darstellung im Browser verantwortlich ist.
Der *AnnotationCoordinator* besteht, wie zuvor erwähnt, aus einer TypeScript und einer *.hbs*-Datei. Die TypeScript-Datei beinhaltet die Steuerung für die Drag&Drop-Bewegung des Fensters. Außerdem wird in dieser der Hover-Effekt je nach Mausbewegung verwaltet. Dies ist ähnlich zu der *PopupCoordinaterComponent* gehalten, da diese das Verhalten bereits für die Pop-ups implementiert hatte. Der *AnnotationCoordinator* beinhaltet zudem die Methode *ping*, welche die vorhandene Ping-Funktion von ExplorViz für ein zugehöriges Objekt aufruft.
Die *.hbs*-Template-Datei der Komponente ist für die Darstellung der Fenster verantwortlich, die im HTML-Format ist. Ember.js bietet dabei die Möglichkeit über simple bedingte Anweisungen die Darstellung zu beeinflussen. Annotationen werden so lediglich im HTML-Canvas gezeichnet, wenn das Attribut *wasMoved* des *AnnotationData*-Objekts wahr ist. Andernfalls wird die Annotation als leerer Div-Container gezeichnet. Das Attribut *isAssociated* wird auf gleiche Art für die Entscheidung genutzt, ob eine freie oder eine zugehörige Annotation gezeichnet werden muss. Auch die Anzeige des Besitzers und des letzten Bearbeiters sowie des zugehörigen Objekts werden über die jeweiligen Attribute angezeigt. Mit dem Attribut *inEdit* wird unterschieden, ob für den Titel ein Label oder ein Eingabefeld gezeichnet wird und ob für den Text die Textarea-Komponente mit der *readonly*-Markierung gekennzeichnet ist. Außerdem ist von diesem Attribut auch die Anzeige des Buttons abhängig, wie zuvor in Abbildung 4.3 gezeigt. Da die Attribute für den Text und den Titel mit *@Tracked* markiert sind, werden diese bei jeder Veränderung durch eine Benutzereingabe, in den jeweiligen Eingabefeldern, automatisch aktualisiert. Bei der Erstellung durch die *BrowserRendering*-Komponente werden den Annotationen außerdem Methoden aus dem *BrowserRendering* und dem *AnnotationHandler* übergeben, welche als Eventziel einem Button übergeben werden. Dies umfasst Methoden zum Minimieren, Öffnen, und Verlassen des Editiermodus sowie zum Teilen und Löschen der Annotation.

4.2.4. Minimierungsfunktion

Wie zuvor bereits erwähnt, ist es in ExplorViz möglich, Annotationen zu minimieren, um so die Übersichtlichkeit zu bewahren. Deshalb wird bereits in der *addAnnotation*-Methode geprüft, ob eine potenzielle Annotation bereits minimiert existiert. Aufgrund der beiden Annotationsarten, die wir für ExplorViz erstellt haben, unterscheidet sich die Minimierungsfunktion in ihrem Vorgehen abhängig von der jeweiligen Annotation.
Listing 4.2 zeigt die Methode *minimizeAnnotation*, welche zur Minimierung von zugehörigen Annotationen genutzt wird. Die Methode sucht über die übergebene ID die Annotation aus der Liste *annotationData* und löst, falls diese im Editiermodus ist, zuvor erwähnte Fehlermeldung aus. Ansonsten entfernt die Methode den Hover-Effekt, falls dieser aktiv ist, von dem zugehörigen Objekt. Anschließend wird das Attribut *wasMoved* auf *false* gesetzt und die Annotation an *minimizedAnnotations* angehängt, während sie aus *annotationData* entfernt wird.

Listing 4.2. Methode zur Minimierung von Annotationen im *AnnotationHandler*.

```
minimizeAnnotation(annotationId: number) {
  const annotation =
      this.annotationData.find((an) => an.annotationId === annotationId);

  if (annotation) {
    if (annotation.inEdit) {
      this.toastHandlerService.showErrorToastMessage('Please␣exit␣edit␣mode␣before
          ␣minimizing.');
      return;
    }

    if (annotation.entity) {
      const mesh = this.applicationRenderer.getMeshById(annotation.entity.id);
      if (mesh?.isHovered) { mesh.resetHoverEffect(); }
    }
    annotation.wasMoved = false;

    this.minimizedAnnotations = [...this.minimizedAnnotations, annotation];
    this.annotationData =
        this.annotationData.filter((an) => an.annotationId !== annotationId );
  }
}
```

Das Minimieren für freie Annotationen sucht ebenfalls zunächst die passende Annotation aus *annotationData* heraus und gibt die gleiche Fehlermeldung aus, falls diese im Editiermodus ist. Dies passiert in der Methode *hideAnnotation* des *AnnotationHandler*. Da die freien Annotationen jedoch lediglich in der Ansicht verkleinert werden, indem die Texteingabe des Fensters eingeklappt wird, wird hier nur das Attribut *hidden* geschaltet. Ist dieses bereits *true*, so wird es auf *false* gesetzt und das Fenster wird in voller Größe angezeigt. Ist es *false*, so wird es auf *true* gesetzt und das Fenster klappt ein.

Damit vor allem die minimierten Annotationen nicht vergessen werden, wird bei Erstellung einer zugehörigen Annotation das Label des jeweiligen Objekts um den Zusatz *'[annotated]'* erweitert. Hierfür haben wir die Klasse *ApplicationRenderer*, die für die Darstellung der *three.js*-Objekte zuständig ist, um die Methode *updateLabel* erweitert. Die Methode bekommt die ID des Objekts und den Labelzusatz übergeben. Über die ID wird zunächst die Referenz auf das 3D-Objekt der Anwendung erhalten, worüber dann das *BoxMesh* des Objekts erhalten werden kann. Dieses wird benötigt, um auf das Label zuzugreifen. Durch die neue Funktion *updateBoxTextLabel* kann dann das Label des Objekts durch die aktualisierte Variante ersetzt werden. Anschließend wird das 3D-Objekt aktualisiert.
Die *updateLabel*-Methode wird im *AnnotationHandler* beim Hinzufügen und Löschen von

Annotationen aufgerufen. Hierbei wird überprüft, dass das Datenmodell des Attributs *mesh* nicht vom Typ *ClazzCommuMeshDataModel* ist, da es sich ansonsten um eine Kommunikationslinie handelt, die kein Label besitzt.

4.2.5. Kollaborationsfunktion

Für die Implementierung der Kollaborationsfunktion für die Annotationen wurde neben dem Frontend auch der Collaboration-Service, der als Websocket-Server für die Kollaborationsfunktion von ExplorViz dient, erweitert. Der Collaboration-Service nutzt dafür die Pub/Sub-Channel von Redis.

Modell für Annotationen im Collaboration-Service
Für die Annotationen haben wir auf Serverseite das Modell *AnnotationModel* erstellt, wie Abbildung 4.9 zeigt. Dieses unterscheidet sich von dem Modell im Frontend, da für die Funktionen der Kollaboration nicht alle bzw. andere Daten notwendig sind. Während die *annotationId* im Frontend die eindeutige ID einer Annotation ist, ist dies im Collaboration-Service die *menuId*. Das Modell erbt von der Klasse *ScalableBaseModel* und implementiert die Schnittstelle *GrabbableObjectModel*. Diese ermöglichen die Zuordnung und das Abrufen von Positionsdaten für Objekte. Neben dem Modell haben wir noch den *AnnotationModifier* erstellt. Dieser besteht unter anderem aus einem *GrabModifier*, der Funktionen wie Löschen und Bewegen für *GrabbableObject*s besitzt, wozu auch die Annotation gehört. Daneben hat der *AnnotationModifier* einen Schlüssel-Wert-Speicher, der als Schlüssel einen String und als Wert ein *AnnotationModel* erwartet und bietet Methoden zum Erstellen und Schließen von Annotationen. Der Collaboration-Service speichert für jeden Kollaborationsraum einen *AnnotationModifier* über den auf die Annotationen des Raums zugegriffen werden kann.

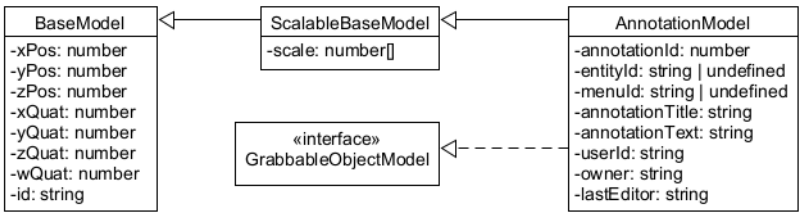

Abbildung 4.9. Klassendiagramm für das Datenmodell der Annotationen im Collaboration-Service.

Raumerstellung und Beitritt
Wenn ein Benutzer einen Kollaborationsraum öffnet, dann werden die Objekte des Raumes serialisiert, damit diese in einem geeigneten Format für die Kommunikation über die WebSocket-Vebindung sind. Aktualisierungen zu den jeweiligen Objekten werden in diesem Format an die anderen Teilnehmer des Raumes gesendet. Auch die Annotationen haben dementsprechend ein serialisiertes Format. Die *SerializedAnnotation* hat einen ähn-

lichen Aufbau wie das *AnnotationModel* des Collaboration-Server zuvor, beinhaltet aber zusätzlich Informationen über den Editierzustand, ob die Annotation geteilt wurde und eine mögliche ID des Objekts im Kollaborationsraum. Für die Raumerstellung haben wir uns an den vorhandenen Pop-ups orientiert, die beim Öffnen eines Raumes geschlossen werden und stattdessen neu erstellt werden müssen. Somit kann verhindert werden, dass ein Nutzer, der bereits in der gleichen Landscape ist und Annotationen geöffnet hat, dem Raum beitritt und so eventuell zwei Annotationen für das gleiche Objekt besitzt.
Tritt ein Benutzer einem bestehenden Raum bei, wird der im Collaboration-Service verwaltete Raum in seiner serialisierten Fassung mit dem neuen Teilnehmer synchronisiert. Bei dieser Synchronisierung wird die *onRestoreAnnotations*-Methode des *AnnotationHandler* getriggert. Diese bekommt die vom Collaboration-Server gesendete Liste der serialisierten Annotationen und erstellt aus jeder davon mit der *addAnnotation*-Methode eine Annotation im Frontend des Teilnehmers. Dabei kann über das Attribut *entityId* das Mesh des zugehörigen Objekts bezogen werden, sofern die ID nicht undefiniert ist.

Teilen von Annotationen
Um eine Annotation mit allen Teilnehmern zu teilen, kann ein Benutzer auf den Teilen-Button der Annotation klicken. Hierdurch wird die Methode *shareAnnotation* des *AnnotationHandler*s aufgerufen. Die Methode sendet in dieser eine Nachricht für das *ANNOTATION_OPENED_EVENT* an den Collaboration-Service. Die Nachricht für dieses Event ist ähnlich zum *AnnotationModel* des Collaboration-Service, wie in Listing 4.3 zu sehen, unterscheidet sich aber darin, dass diese einen Eventtypen und eine Nonce besitzt und mit *inEdit* einen Wert hat, der den Zustand des Editiermodus angibt. Die Nonce ist eine einmalige Nummer, die dafür benutzt wird, die Antwortnachricht zuzuordnen.

Listing 4.3. Aufbau der Nachricht für ein ANNOTATION_OPENED_EVENT.

```
export type AnnotationOpenedMessage = {
  event: typeof ANNOTATION_OPENED_EVENT;
  nonce: Nonce;
  annotationId: number;
  entityId: string | undefined;
  menuId: string | null;
  annotationTitle: string;
  annotationText: string;
  owner: string;
  inEdit: boolean;
  lastEditor: string;
};
```

Wurde die Nachricht vom Websocket-Client im Frontend verschickt, empfängt der Websocket-Server diese, wodurch die Methode *handleAnnotationOpenedMessage* des Websocket-Gateways getriggert wird. In dieser wird eine ID generiert, die als die *objectID*, also dem Schlüssel

für den Schlüssel-Wert-Speicher, des *AnnotationModifiers* gesetzt wird. Mit der ID und der Nachricht wird dann eine *RoomForwardMessage* erstellt, die für das Verteilen an die Teilnehmer des Raumes gebraucht wird. Diese Verteilung wird zuletzt getriggert und am Ende wird dem initiierenden Client eine Nachricht für das *ANNOTATION_REPONSE_EVENT* gesendet. Diese beinhaltet neben der Nonce zum Identifizieren die generierte ID. Nach Erhalt der Antwort wird im Frontend des Clients das Attribut *sharedBy* auf den aktuellen Benutzer, das Attribut *shared* auf *true*, und das Attribut *menuId* auf die empfangene *objectId* gesetzt.
Die Verteilung der zuvor erstellten Raumnachricht wird von der Methode *handleAnnotationEvent* des *SubscriberService* im Collaboration-Service übernommen. Diese erstellt zunächst für den jeweiligen Raum eine neue Annotation im *AnnotationModifier*. Anschließend wird dann eine Weiterleitungsnachricht an alle anderen Teilnehmer des Raumes gesendet. Der Aufbau der Nachricht entspricht dem der serialisierten Annotation. Empfängt ein Client diese Nachricht triggert dies die Methode *onAnnotation* des *AnnotationHandler*s, wodurch eine Annotation für die serialisierte Annotation in der Landscape erstellt wird.

Editieren und Aktualisieren von Annotationen
Damit eine Annotation verändert und aktualisiert werden kann, gibt es neben dem Ansichtsmodus auch den Editiermodus. In diesem ist es außerhalb eines Kollaborationsraumes immer möglich zu wechseln. Innerhalb eines Kollaborationsraumes ist dies anders. Klickt der Benutzer auf die Schaltfläche, um in den Editiermodus zu wechseln, wird eine Nachricht mit einem *ANNOTATION_EDIT_EVENT* an den Collaboration-Service gesendet. Diese beinhaltet die *menuID*, respektive die *objectId* der Annotation, den Eventtypen, und eine Nonce. Sobald der Collaboration-Service die Nachricht empfangen hat, wird die Methode *handleAnnotationEditMessage* des Gateways getriggert. Diese bezieht über den Client den Raum, um den es sich handelt, um anschließend darüber an das *GrabbableObject*, also die Annotation, zu kommen. Mittels eines *Redlocks*[1] wird dann versucht das Objekt in dem Raum für den Benutzer zu sperren. Somit ist ein gegenseitiger Ausschluss gesichert und es kommt zu keinen Problemen, in denen mehrere Benutzer gleichzeitig die gleiche Annotation verändern möchten. Wird das Objekt für den Benutzer gesperrt, kann eine Antwortnachricht an diesen versendet werden, die einen Parameter *isEditable* auf *true* gesetzt hat. Kann das Objekt nicht gesperrt werden, da ein anderer Benutzer dieses gesperrt hat, wird eine Antwortnachricht mit *isEditable* auf *false* gesendet. Je nach Art der Antwort wird bei Empfang im Client entweder der Editiermodus, durch Setzen des Attributs *inEdit* geöffnet oder die in Abschnitt 4.1.3 erwähnte Fehlermeldung angezeigt.

Listing 4.4. Aufbau der Nachricht für ein ANNOTATION_UPDATED_EVENT.

```
1 export type AnnotationUpdatedMessage = {
2   event: typeof ANNOTATION_UPDATED_EVENT;
3   nonce: Nonce;
```

[1] https://redis.io/docs/latest/develop/use/patterns/distributed-locks/

```
4   objectId: string;
5   annotationId: number;
6   annotationTitle: string;
7   annotationText: string;
8   lastEditor: string;
9 };
```

Hat der Benutzer seine Änderungen im Editiermodus gemacht, bestätigt er dieses durch Klicken der Schaltfläche '*Update Annotation*'. Dadurch wird der Editiermodus verlassen und die Methode *updateAnnotation* im *AnnotationHandler* aufgerufen. Sollte es sich nicht um einen Kollaborationsraum handeln, wird in diesem Fall nur das Attribut *lastEditor* geändert und *inEdit* auf *false* gesetzt. Anderenfalls wird eine Nachricht für ein *ANNOTATION_UPDATED_EVENT*, dessen Aufbau in Listing 4.4 zu sehen ist, an den Collaboration-Service gesendet. Hierbei entspricht die *objectId* der *menuId* der Annotation im Frontend. Durch Empfang der Nachricht wird im Gateway die Methode *handleAnnotationUpdatedMessage* getriggert, die mithilfe der übergebenen *objectId* eine Raumnachricht erstellt. Anschließend wird die Verteilung der Raumnachricht über den *SubscriberService* gestartet und das Redlock für das Objekt wieder freigegeben. Zuletzt wird eine Antwortnachricht mit dem Attribut *updated*, das wahr ist, an den initiierenden Client gesendet. Dieser setzt bei Erhalt das *inEdit*-Attribut auf *false*, sofern *true* empfangen wurde. Ansonsten wird eine Fehlermeldung angezeigt.
Die Verteilung der Raumnachricht wird durch die Methode *handleAnnotationUpdatedEvent* des *SubscriberService*s abgehandelt. In dieser wird zunächst mit Hilfe der *objectId* die richtige Annotation im *AnnotationModifier* ermittelt. Anschließend werden in dieser die Attribute *annotationTitle*, *annotationText*, und *lastEditor* mit dem empfangenen Wert aktualisiert. Die aktualisierten Attribute werden zuletzt zusammen mit der *objectId* als Weiterleitungsnachricht an alle anderen Teilnehmer des Raumes versendet. Nach dem Empfang der Weiterleitungsnachricht im Client wird ähnlich wie im Collaboration-Service, über die getriggerte Methode *onUpdatedAnnotation* im *AnnotationHandler*, die richtige Annotation ermittelt und die jeweiligen Attribute aktualisiert. Da diese Attribute alle die Markierung *@Tracked* besitzen, ist ein Neuladen des Canvas bzw. des Objekts nicht nötig, da diese automatisch neugezeichnet werden.

Löschen von Annotationen
Um Annotationen aus der Landscape zu entfernen, gibt es auch eine Schaltfläche zum Löschen, die beim Klicken die Methode *removeAnnotation* des *AnnotationHandler*s aufruft. Außerhalb eines Kollaborationsraumes wird hierbei das Label zurückgesetzt, die Hover-Effekte entfernt, und schließlich die Annotation aus der Liste *annotationData* entfernt. Befindet sich der Benutzer in einer Kollaborationssitzung, wird davor jedoch geprüft, ob die Annotation fehlerfrei gelöscht werden kann. Hierfür wird die Methode *canRemoveAnnotation* des *AnnotationHandler*s aufgerufen, die einen boolschen Wert zurückgibt. In der Methode wird eine Nachricht für ein *ANNOTATION_CLOSED_EVENT*, die den Eventtypen, die *me-*

nuId, und eine Nonce enthält, an den Collaboration-Service versendet. Durch den Empfang dieser Nachricht wird im Gateway des Servers die Methode *handleAnnotationClosedMessage* getriggert. Diese versucht, wie zuvor beim Aktualisieren erklärt, das Objekt der Annotation für den anfragenden Benutzer zu sperren. Ist dies nicht möglich, so wird eine Antwortnachricht mit dem Wert *false* für den Parameter *isSuccess* an den initiierenden Client gesendet, woraufhin dieser die Fehlermeldung *'Could not remove popup since it is currently in use by another user.'* anzeigt und die Annotation nicht gelöscht wird. Ist es möglich, das Objekt zu sperren, wird über den *SubscriberService* mit der Methode *handleAnnotationClosedEvent* die Verteilung einer Weiterleitungsnachricht an alle Teilnehmer gestartet. In dieser wird über die *closeAnnotation*-Methode des *AnnotationModifier* die zugehörige Annotation aus dem Raum entfernt. Anschließend wird die Weiterleitungsnachricht an die Raumteilnehmer versendet. Diese besitzt die ursprüngliche im Collaboration-Service empfangene Nachricht. Im Client wird nach Empfang dieser Weiterleitungsnachricht die Methode *onMenuClosed* des *AnnotationHandler*s getriggert, die mit der empfangenen *menuId* die dazu passende Annotation aus der Liste *annotationData* entfernt und somit auch aus der Landscape.

Kapitel 5

Persistierung

Ein Ziel dieser Arbeit war die Persistierung von Snapshots und benutzerdefinierten Git-Hosting-Token. Im Folgenden erläutern wir den Ansatz, mit dem dieses Ziel erreicht wurde und gehen auf die Implementierung dieses Ansatzes ein.

5.1. Ansatz

5.1.1. Ansatz für die Persistierung von Snapshots

Für die Persistierung von Snapshots haben wir den vorhandenen User-Service erweitert. Dieser bot bereits mit *Quarkus Panache*[1] eine MongoDB-Instanz, die wir um eine Collection für die Snapshots erweitert haben. Die Nutzung einer relationalen Datenbank wäre hierfür machbar, da die Daten des Benutzers, der Snapshots, und der in Abschnitt 5.1.2 folgenden benutzerdefinierten Git-Hosting-Token gut über Relationen darstellbar sind. Hiergegen haben wir uns aufgrund der vorhandenen MongoDB-Instanz und der Tatsache, dass die zu speichernden Daten keine komplexen Abhängigkeiten voneinander haben, entschieden. Listing 5.1 zeigt dessen Datenbankschema. Der Datentyp *Document* beschreibt ein BSON-Dokument für die MongoDB, welche eine binäre Repräsentation von JSON-Dokumenten ist und zusätzlich mehr Datentypen enthält. Die Implementierung im User-Service verwaltet die Snapshots und bietet eine Schnittstelle, damit das Frontend mit diesem kommunizieren kann. Im Frontend wurde die Implementierung der Snapshots im Zuge der Bachelorarbeit von Julius Brehme [Brehme 2024] umgesetzt, in welcher genauer auf die Zusammenstellung des Snapshots eingegangen wird.
Es ist möglich Snapshots zu persistieren und zu löschen. Das Aktualisieren eines vorhandene Snapshots ist nicht möglich. Wir sind der Meinung, dass so das Ziel der Snapshots, dass diese zum Teilen einer Ansicht genutzt werden, verfehlt werden würde. In einem Snapshot werden alle nötigen Daten gespeichert, um bei Aufruf von diesem die gesamte Landscape so zu laden, wie sie gespeichert wurde. Dabei unterscheiden wir bei der Anfrage für die Snapshots, ob ein Snapshot geladen werden soll oder für die Landscapeübersichtsseite angefragt wird. In letzterem Fall werden die Snapshots nicht mit allen Daten versendet, sondern nur mit den nötigsten. So wird ein unnötig großer Datenverkehr vermieden.
Außerdem gibt es die Möglichkeit, dass Snapshots automatisch gelöscht werden, sobald

[1] https://quarkus.io/guides/hibernate-reactive-panache

ein angegebenes Datum erreicht wurde. Ein Snapshot kann zudem geteilt werden. Für diesen Zweck verfügt jeder Snapshot über ein JSON-Attribut, in welchem die Benutzer gespeichert werden, die einen geteilten Snapshot geöffnet und somit 'abonniert' haben. Diese werden im Folgenden als Subscriber bezeichnet. Wir unterscheiden zwischen geteilten und persönlichen Snapshots. Wird ein Snapshot geteilt, so wird eine Kopie von diesem angelegt. Dies ermöglicht es, den geteilten Snapshot zu löschen, ohne den persönlichen zu verlieren. Die Subscriber haben die Möglichkeit einen abonnierten Snapshot zu deabonnieren.

Listing 5.1. Das MongoDB Schema für die Snapshot-Collection für eine einzelne gespeicherte Landscape und der zugehörigen Darstellung.

```
{
    owner: String,
    createdAt: Long,
    name: String,
    landscapeToken: Document,
    structureData: Document,
    serializedRoom: Document,
    timestamps: Document,
    camera: Document,
    isShared: boolean,
    subscribedUsers: Document,
    deleteAt: Long
}
```

5.1.2. Ansatz für die Persistierung von benutzerdefinierten Git-Hosting-Token

Wie auch für die Snapshots haben wir für die Persistierung der benutzerdefinierten Git-Hosting-Token den User-Service erweitert. Hierfür haben wir eine weitere Collection erstellt, dessen Schema Listing 5.2 zeigt.

Listing 5.2. Das MongoDB Schema für die UserApi-Collection für benutzerdefinierte Git-Hosting-Token.

```
{
    uid: String,
    name: String,
    token: String,
    hostUrl: String,
    createdAt: Long,
    expires: Long
}
```

Die Implementierung im User-Service verwaltet benutzerdefinierte Git-Hosting-Token, im Folgenden *UserApi-Token* und bietet eine Schnittstelle, damit das Frontend mit diesem kommunizieren kann. Es ist möglich neue UserApi-Token anzulegen und bestehende zu lesen und zu löschen. Außerdem ist über das Setzen eines Auslaufdatums das automatische Löschen der UserApi-Token, ähnlich wie bei den Snapshots, möglich. Wie bei den Snapshots wurde die Implementierung der benutzerdefinierten Git-Hosting-Token im Frontend im Zuge der Bachelorarbeit von Julius Brehme [Brehme 2024] beschrieben.

5.2. Implementierung

Für die Implementierung der Persistierung erweitern wir den User-Service. Die Daten werden in einer MongoDB mit Panache gespeichert. Über einen Quarkus JSON REST Service werden die zu persistierenden Daten übermittelt und abgerufen.

5.2.1. Snapshot

Die Klasse der Snapshots besitzt die im Schema in Listing 5.1 gezeigte Struktur. Über die Klasse wird die Collection *snapshot* gebildet, in welcher die Dokumente der Snapshots gespeichert werden. Die Implementierung folgt dem Repository-Muster[2], verfügt also über eine Klasse *SnapshotRepository*, welche das Interface *PanacheMongoRepositoryBase* erweitert. Mit diesem definieren wir die Klasse *Snapshot* als die Entitätsklasse der Collection. Das Interface bietet außerdem bereits viele Basisoperationen, wie zum Beispiel das Löschen eines Eintrages. Wir erweitern die Repository-Klasse um drei Methoden:

- *findForUser*, welche den Besitzer als String übernimmt und eine Liste aller Snapshots mit diesem Besitzer zurückgibt
- *findForUserAndCreatedAtAndIsShared*, welche den Besitzer, das Erstelldatum, und den booleschen Wert, ob der Snapshot geteilt wurde, erhält und eine Liste mit allen dazu passenden Snapshots zurückgibt
- *getAll*, welche eine Liste aller Snapshots zurückgibt

Dabei ist zu erkennen, dass als eindeutige Attribute der Besitzer, also *owner* und das Erstelldatum, *createdAt*, genutzt werden. Das *isShared*-Attribut ist wichtig, um zu unterscheiden, ob es sich um einen persönlichen oder einen geteilten Snapshot handelt. Mit dem Attribut *subscribedUsers* besitzt ein Snapshot eine Liste über alle Benutzer, die einen geteilten Snapshot geöffnet haben. Dies ist so umgesetzt, dass bei Erstellung eines geteilten Snapshots eine URL im Frontend erzeugt wird, bei der der öffnende Benutzer der Liste hinzugefügt wird, sofern dieser nicht bereits in der Liste vorhanden ist.

[2] https://quarkus.io/guides/mongodb-panache#solution-2-using-the-repository-pattern

5. Persistierung

REST-Schnittstelle
Für die Kommunikation zwischen dem Frontend und dem User-Service haben wir eine REST-Schnittstelle implementiert. Diese kann auch für zukünftige Anwendungszwecke genutzt werden. Damit alle nötigen Operationen wie Lesen, Erstellen, und Löschen stattfinden können, verfügt die Schnittstelle über folgende Anfragepunkte:

- *createNewSnapshot*, die mittels einer POST-Anfrage einen übermittelten Snapshot persistiert
- *deleteSnapshot*, die mittels einer DELETE-Anfrage einen Snapshot löscht, welcher sich mit den übermittelten Attributen *owner*, *createdAt*, und *isShared* gleicht
- *getSnapshotByValue*, die über eine GET-Anfrage alle Snapshots in minimierter Struktur zurückgibt, die als persönliche oder geteilte Snapshots des übermittelten Besitzers gespeichert sind
- *getSnapshot*, die über eine GET-Anfrage, die die Attribute *owner*, *createdAt*, *isShared*, und *subscriber* liefert, den gesuchten Snapshot zurückgibt. Hierbei wird der *subscriber* an die Liste *subscribedUsers* angehangen, sofern dieser ungleich dem Besitzer ist und *isShared* auf *true* ist
- *subscribeSnapshot*, die über eine PUT-Anfrage wie auch *getSnapshot* einen Benutzer zu der *subscribedUsers*-Liste eines Snapshots hinzufügen kann, allerdings nichts zurückgibt
- *unsubscribeSnapshot*, die über eine PUT-Anfrage einen übermittelten Benutzer aus der Liste *subscribedUsers* eines geteilten Snapshots entfernt, der dem empfangenen Besitzer und Erstelldatum entspricht
- *shareSnapshot*, die über eine PUT-Anfrage die Attribute *owner*, *createdAt*, und *deleteAt* erhält und über diese einen persönlichen Snapshot als geteilten Snapshot mit dem Löschdatum kopiert

Die Anfragen für das Erstellen, Löschen, oder das Teilen geben nichts an das Frontend zurück. Da in diesen Fällen allerdings Fehler auftreten können, zum Beispiel weil kein passender Snapshot gefunden wurde, werden unterschiedliche HTTP-Statuscodes zurückgesendet, die im Frontend verwaltet werden müssen.
Da für die Anzeige der Landscapeübersichtsseite nicht alle Informationen der Snapshots gebraucht werden, haben wir uns entschieden, für diese eine minimierte Variante zu verschicken. Dies passiert in der Methode *getSnapshotByValue*. In dieser wird über alle Snapshots der Collection iteriert. Die Snapshots werden anschließend in eine der drei Listen *personalSnapshots*, *sharedSnapshots*, oder *subscribedSnapshots* verteilt. Die Liste, in welche der Snapshot kommt, ist abhängig davon, ob der anfragende Benutzer der Besitzer ist und dann abhängig von dem Wert des *isShared*-Attributs oder ob der anfragende Benutzer in der *subscribedUsers*-Liste eines Snapshots vorkommt. Die minimierten Snapshots, die für die Anzeige auf der Übersichtsseite benötigt werden, umfassen die Attribute *owner*,

createdAt, *name*, und *landscapeToken*. Es werden somit die Attribute eingespart, die durch die Beschreibung der Strukturen der Landscape verhältnismäßig groß werden können. Wurden alle betroffenen Snapshots in die jeweiligen Listen sortiert, wird ein JSON mit den drei Listen an den Anfragenden gesendet.

Automatisches Löschen
Über das Attribut *deleteAt* kann einem Snapshot mitgegeben werden, wann dieses automatisch aus der Datenbank entfernt werden soll. Anstatt, dass auf dem Server eine stetige Routine läuft, die die Snapshots auf das Erreichen des Datums prüft, haben wir uns dafür entschieden, dies semi-automatisch umzusetzen. Da beim Öffnen von ExplorViz immer die Anfrage gesendet wird, die die zuvor bereits besprochene *getSnapshotByValue*-Methode triggert, haben wir diese so implementiert, dass hierbei das Löschdatum mit überprüft wird. Ist dies erreicht, so wird der Snapshot aus der Datenbank gelöscht. Da die Methode aufgrund der Subscriberlisten über alle Snapshots iteriert, betrifft dies auch Snapshots, die mit dem anfragenden Benutzer nichts zu tun haben. Dieser Ansatz spart dauerhaft Ressourcen auf Kosten von eventuell längeren Ladezeiten für die Überprüfung bei der Anfrage.

5.2.2. UserApi

Für die benutzerdefinierten Git-Hosting-Token haben wir die Klasse *UserApi* erstellt, welche der Struktur aus Listing 5.2 folgt. Wie zuvor bei den Snapshots, wird hierfür die MongoDB Collection *userapi* gebildet und es wird erneut das Repository Muster angewandt. Dementsprechend haben wir das Interface *PanacheMongoRepositoryBase* um die Klasse *UserApiRepository* erweitert, welche die Klasse *UserApi* als Entitätsklasse verwendet. Diese Klasse erweitert die vorhandenen Repositorymethoden um die folgenden Methoden:

- *findForUser*, welche für eine gegebene Benutzer-ID eine Liste aller passender *UserApi*-Objekte aus der Datenbank zurückgibt
- *findForUserAndToken*, welche für eine gegebene Benutzer-ID und einen gegebenen Git-Hosting-Token eine Liste von dazu passenden *UserApi*-Objekten aus der Datenbank zurückgibt

Für weitere benutzerspezifische Einstellungen und Objekte haben wir die *UserApi*-Klasse konzipiert, damit diese zukünftig für diese Fälle erweitert werden kann. Die Klasse kann somit als Ausgangspunkt für weitere Benutzereinstellungen genutzt werden.

REST-Schnittstelle
Wie zuvor haben wir auch für die benutzerdefinierten Git-Hosting-Token eine REST-Schnittstelle implementiert, damit zwischen dem Frontend und dem User-Service kommuniziert werden kann. Da für die Token nicht viel Logik benötigt wird, verfügt die Schnittstelle über die folgenden, simplen Anfragepunkte:

- *createNewUserApi*, welche mittels einer POST-Anfrage alle benötigten Attribute empfängt und daraus ein neues *UserApi*-Objekt erstellt und persistiert
- *deleteUser*, welche mittels einer DELETE-Anfrage eine Benutzer-ID und einen Git-Hosting-Token erhält und den dazu passenden *UserApi*-Eintrag aus der Datenbank entfernt
- *getUserByValue*, welche mittels einer GET-Anfrage eine Benutzer-ID erhält und darüber eine Liste aller zugehörigen *UserApi*-Objekte zurückgibt

Um Fehler und Probleme mit dem Frontend zu kommunizieren, versenden wir verschiedene HTTP-Statuscodes, wenn zum Beispiel bereits ein Objekt für die übergebene Benutzer-ID und den Git-Hosting-Token existiert oder ein zu löschendes Objekt nicht gelöscht werden konnte.

Automatisches Löschen
Wie auch bei den Snapshots, können benutzerdefinierte Git-Hosting-Token automatisch aus der Datenbank entfernt werden. Hierfür wird bei der Erstellung das Attribut *expires* gesetzt. Anders als zuvor gibt es für die *UserApi*-Objekte keine Anfrage, in der über alle vorhandenen Objekte iteriert wird. Aus diesem Grund wird für die Git-Hosting-Token die gleiche Logik genutzt, dass bei der GET-Anfrage überprüft wird, ob das Datum erreicht wurde und wenn dies der Fall ist, der Token gelöscht wird. Allerdings passiert dies nur bei den vom anfragenden Benutzer eigenen Token.

Kapitel 6

Evaluation

In diesem Abschnitt präsentieren wir die Evaluation der von uns entwickelten Funktionserweiterungen für ExplorViz in Form einer Usability-Studie. Aufgrund der thematischen Nähe wurde die Evaluation gemeinsam mit Julius Brehme für dessen Bachelorarbeit [Brehme 2024] durchgeführt.

6.1. Ziele

Im Zuge der Evaluation der implementierten Funktionen wollen wir herausfinden, welche Potenziale und Mängel diese besitzen. Um dieses Ziel zu erreichen, spezifizieren wir drei Forschungsfragen, welche die allgemeine Benutzbarkeit der Funktionen offenlegen sollen:

- Wie sinnvoll sind die implementierten Funktionen in ExplorViz?
- Wie zuverlässig sind die implementierten Funktionen?
- Wie verständlich ist das Design der implementierten Funktionen?

6.2. Methodik

Um die Potenziale und Mängel unserer implementierten Funktionen herauszufinden, führen wir eine Usability-Studie durch [Plaisant 2004]. Um außerdem herauszufinden, wie sinnvoll unsere implementierten Funktionen sind, haben wir uns zusätzlich für eine schriftliche Befragung entschieden. Die Probanden bekommen zur Evaluation einen Online-Fragebogen mit geschlossenen Fragen, welche vorgegebene Antworten besitzen und offenen Fragen, die frei zu beantworten sind. Der Fragebogen enthält zusätzlich eine schriftliche Verfassung der Aufgaben, die die Probanden vor der Beantwortung der jeweiligen Fragen durchführen sollen. Die Aufgaben dienen dazu, die Probanden mit den Funktionen vertraut zu machen und so die Stärken und Schwächen dieser erkennbar zu machen.

6.3. Experiment

Zur Evaluation unseres Ansatzes führen wir eine Usability-Studie durch. Im Folgenden präsentieren wir den hierfür genutzten Versuchsaufbau, der aus den Probanden und der

genutzten Systeme besteht und erläutern den Ablauf der Evaluation. Da die Evaluation zusammen mit den Inhalten der Bachelorarbeit von Julius Brehme [Brehme 2024] durchgeführt wurde, enthalten der Versuchsaufbau und der Fragenbogen Inhalte, die für diese Arbeit nicht von Relevanz sind, wie zum Beispiel die GHS-Facade als Dienst. Im restlichen Verlauf des Abschnitts wird deshalb lediglich auf die für unsere Arbeit relevanten Inhalte eingegangen.

6.3.1. Versuchsaufbau

Um die Potenziale und Mängel unserer implementierten Funktionen herauszufinden, benötigten wir eine ExplorViz-Instanz, die aus den Diensten Frontend, User-Service, Collaboration-Service, und GHS-Facade besteht. Da die implementierten Funktionen nur bedingt abhängig von der genutzten Landscape sind, können wir den Demo-Supplier von ExplorViz nutzen, der bereits vordefinierte Landscapes enthält. In der aktuellen Entwicklungsumgebung von ExplorViz wird der Dummy-Benutzer *Johnny* verwendet. Damit die Probanden ein Gefühl für die Funktionalität der Snapshots, vor allem für das Teilen dieser, bekommen, haben wir im Vorhinein den Dummy-Benutzer zu einem anderen geändert. Mit diesem haben wir anschließend, über die Git-Hosting-Anbindung, ein Issue in GitLab mit einem Snapshot erstellt. Danach haben wir den Dummy-Benutzer wieder auf *Johnny* zurückgesetzt. Dadurch können die Probanden einen Snapshot eines anderen Benutzers über einen Link öffnen.

Probanden
ExplorViz ist eine Software-as-a-Service Anwendung zur Visualisierung von Softwaressystemen. Aus diesem Grund sind Personen mit einem Hintergrund in der Informatik unsere primäre Zielgruppe für die Evaluation. Da die Benutzung der implementierten Funktionen allerdings keine Kenntnisse der Informatik erfordern, bilden Personen ohne Informatik-Hintergrund eine sekundäre Zielgruppe. Wir sind auf freiwillige Probanden angewiesen und geben diesen keine finanziellen Anreize. Um das Ungleichgewicht zwischen den unterschiedlichen Wissensständen der Probanden in Bezug auf ExplorViz und Softwarevisualisierungen auszugleichen, geben wir eine Einführung, die je nach Kenntnisstand in der Ausführlichkeit variiert.

Ausrüstung
Für die Durchführung der Evaluation werden zwei Laptops mit jeweils einem zusätzlichen Monitor mit einer Auflösung von 1920x1080 und einer Größe von 24 bzw. 27 Zoll, die über HDMI angeschlossen werden, genutzt. Dadurch besteht die Möglichkeit, dass zwei Probanden simultan evaluiert werden. Der zweite Monitor ermöglicht es den Probanden, auf einem Monitor ExplorViz zur Ausführung der Aufgaben zu haben, während auf dem anderen Monitor der Online-Fragebogen offen ist. Zudem können die Probanden die zwei Monitore für die Aufgaben zu der Kollaborationsfunktion nutzen. Die Tabelle 6.1 zeigt die Konfigurationen der beiden Systeme.

Tabelle 6.1. Konfiguration der Systeme, die bei der Evaluation genutzt wurden.

	Laptop 1	**Laptop 2**
Prozessor	Intel® Core™ i7-8550U 1.80GHz x 8	Intel Core™ i7-13700H 3.7GHz x 14
Arbeitsspeicher	16,0 GB	32,0 GB
Festplatte	512,0 GB SSD	512,0 GB SSD
Grafik	Nvidia GeForce MX150	Intel Iris Xe Graphics
Systemtyp	64-Bit-Betriebssystem, x64-basierter Prozessor	64-Bit-Betriebssystem, x64-basierter Prozessor
Betriebssystem	Ubuntu 22.04.4 LTS	Ubuntu 22.04.4 LTS
Auflösung	1920x1080	1920x1080

Software-Konfiguration
Beide Systeme laufen mit dem Betriebssystem Ubuntu 22.04.4 LTS, auf dem zusätzlich *Docker*[1] installiert wurde. Docker wird genutzt um zusätzliche Anwendungen wie Redis und Mongo-Datenbanken zu erstellen und auszuführen. Auf beiden Systemen wird für die Darstellung von ExplorViz der Browser *Mozilla Firefox*[2] genutzt. Für die Evaluation werden auf den Systemen das Frontend sowie die anderen ExplorViz-Dienste gestartet und die zusätzlichen Anwendungen über Docker Compose ausgeführt. Zusätzlich wird eine Datei mit Daten, die für die Aufgabe zur Git-Hosting-Anbindung benötigt werden, bereitgestellt und der Online-Fragebogen im Browser geöffnet.

6.3.2. Einführung in ExplorViz

Damit alle Probanden ungefähr die gleichen Grundkenntnisse von ExplorViz haben, gaben wir vor der Durchführung der Evaluation eine Einführung dazu. Für diese haben wir die Demo von ExplorViz, die online[3] verfügbar ist, benutzt. Zu Beginn der Einführung haben wir die Probanden nach ihrem aktuellen Kenntnisstand von ExplorViz gefragt. Davon abhängig gestaltete sich die Ausführlichkeit der Einführung. Hierbei war wichtig, dass die Probanden außerhalb der Landscape eine Übersicht über die Landscapeübersichtsseite und das Benutzermenü haben. In der Landscape war es wichtig, dass es Kenntnisse über die Kollaborationsfunktion, den Restructuring-Modus, und dem Einstellungsmenü, über dass diese beiden Funktionen erreichbar sind, gibt. Außerdem wurde darauf geachtet, dass die Probanden die Navigation und Interaktion, wie Öffnen und Highlighten, mit den Objekten kennen und dass ein grundsätzliches Verständnis von der Navigation mit Pop-ups vorhanden ist.
Den Probanden wurde außerdem eine kurze Erläuterung der neuen Funktionen, die in der

[1] https://www.docker.com/
[2] https://www.mozilla.org/de/firefox/
[3] https://explorviz.dev/

Evaluation behandelt werden, gegeben. Hierbei war unter anderem wichtig, die Begriffe 'freie Annotation' und 'zugehörige Annotation' zu erklären, da diese im Fragebogen vorkommen. Bei diesen Erläuterungen wurde lediglich auf die Funktion eingegangen und wofür diese gedacht ist. Es wurde nicht erläutert, wie die Funktionen zu benutzen sind. Abschließend wurde den Probanden mitgeteilt, dass diese sich während der Aufgaben frei in ExplorViz bewegen können und ihre Ideen ausprobieren dürfen.

6.3.3. Aufgaben

Der bereitgestellte Online-Fragebogen besteht aus sieben Teilen. Die Tabelle 6.2 bietet einen Überblick über diese. Vier von diesen Teilen beinhalten Aufgaben zu verschiedenen Funktionen. Die Teile C, E, und F handeln von Themen, die nur indirekt relevant sind. Diese behandeln vor allem die Benutzbarkeit des Frontends zum Erstellen und Löschen von Token für die Git-Hosting-Anbindung (Teil C), zum Erstellen, Löschen, und Teilen von Snapshots (Teil E), und der Erstellung eines Issues über die Git-Hosting-Anbindung (Teil F).

Tabelle 6.2. Übersicht über die Teile des Online-Fragebogens [Wulff und Brehme 2024].

Teil-ID	Fragenteil	Inhalt
A	Personenbezogene Daten	Beschäftigung und Bildungsabschluss
B	Fähigkeiten	Erfahrungen in verschiedenen Bereichen
C	Benutzerverwaltung	Erstellung von UserApi-Token
D	Annotationen	Umgang mit Annotationen
E	Snapshots	Umgang mit Snapshots
F	Erweiterung der Git-Hosting Anbindung	Erstellen von Issues
G	Allgemeine Rückmeldung	teilunabhängige Rückmeldung

Teil D des Fragebogens handelt hingegen von den Annotationen. In diesem Teil werden die Probanden dazu aufgefordert, zunächst die Landscape *VISSOFT 23 - Study Sample* des Demo-Suppliers von ExplorViz zu öffnen. In dieser soll zuerst eine freie Annotation mit einem Titel und einem Text erstellt werden. Nach dem Erstellen sollen die Probanden die Annotation minimieren und anschließend löschen. Da das Minimieren der freien Annotation nur das Textfeld einklappt, ist das Löschen direkt möglich. Als nächstes sollen die Probanden einen Kollaborationsraum erstellen und diesem über einen zweiten Tab im Browser beitreten. Für das Testen der kollaborativen Funktionen der Annotation wäre eine Durchführung mit mehreren Probanden gleichzeitig durchaus sinnvoll. Wir haben uns dagegen entschieden, da die Funktionen zum einen auch als einzelner Benutzer mit mehreren Tabs benutzbar sind und zum anderen, da dies der einzige Aufgabenteil ist, in dem eine Kollaboration getestet wird und somit die Probanden zeitlich zu sehr voneinander abhängig wären. Nachdem der Proband mit zwei Tabs im Kollaborationsraum ist, soll

dieser eine zugehörige Annotation an einem Paket mit dem Namen 'org' erstellen und diese anschließend mit dem anderen Teilnehmer (dem zweiten Tab) teilen. Anschließend wird der Proband dazu aufgefordert, die Annotation zu minimieren und diese danach erneut zu öffnen. Beim Minimieren von zugehörigen Annotationen verschwinden diese aus der Visualisierung und müssen erneut geöffnet werden. Durch diese Aufgabe kann getestet werden, ob die Probanden durch die Labelerweiterung *'[annotated]'* an dem Paket erkennen, dass die Annotation noch existiert. Danach werden die Probanden dazu aufgefordert, in dem jeweils anderen Tab die Annotation zu editieren, sodass die Änderung für alle Teilnehmer des Kollaborationsraums sichtbar ist. Hierdurch bekommen die Probanden einen Einblick in die Austauschfunktionalität und können die Sperrfunktion des Editiermodus entdecken. Zuletzt sollen die Probanden die Annotation löschen, wodurch diese für alle Teilnehmer der Kollaboration gelöscht wird.
Die Aufgaben decken alle Funktionalitäten ab, zeigen dem Probanden aber nicht unbedingt alle Fälle, die bei der Benutzung auftreten können. Aus diesem Grund wird in der Einführung darauf hingewiesen, dass die Probanden sich während der einzelnen Aufgaben frei in ExplorViz bewegen können und Funktionen und Ideen ausprobieren sollen. Während der gesamten Evaluation war es für die Probanden möglich Fragen zu stellen und Hilfe zu erhalten.

6.3.4. Fragebogen

Ein Teil der Usability-Studie war die Bearbeitung eines Online-Fragebogens [Wulff und Brehme 2024], damit wir Feedback zu den jeweiligen Funktionen erhalten können. Dieser besteht aus sieben Teilen, wobei Teil C, D, E, und F aus jeweils einem Aufgabenblock und einem Fragenblock bestehen. In diesen sollen die Probanden zunächst die jeweilige Aufgabe durchführen und anschließend die dazugehörenden Fragen beantworten, bevor mit dem nächsten Teil begonnen wird.
Der Fragenbogen startet zunächst mit einem Begrüßungstext, der den Probanden den Zweck, den Ablauf, und die Verwendung der Evaluation und ihrer Ergebnisse erklärt. Die Teile A und B behandeln Fragen zu der Person der Probanden. In Teil A geben die Probanden eine Auskunft über ihr Beschäftigungsverhältnis. Hierbei gibt es die Auswahlmöglichkeiten *Student (Informatik)*, *Student (Nicht-Informatik)*, *Wissenschaftlicher Mitabeiter (Informatik)*, *Wissenschaftlicher Mitabeiter (Nicht-Informatik)*, *Mitabeiter in der Informatik*, *Mitarbeiter außerhalb der Informatik*, und *Sonstiges*. Für die Antwortmöglichkeit *Sonstiges* können Probanden ihre Beschäftigung über ein Textfeld angeben. Außerdem werden die Probanden nach ihrem Bildungsabschluss mit der Auswahl aus *Allgemeine Hochschulreife*, *Ausbildung*, *Bachelor*, *Master*, *Doktor*, *Professur*, und *Sonstiges* gefragt. Diese Fragen ermöglichen uns später mögliche Zusammenhänge zwischen den Probanden und der Bewertung der Funktionen aufzudecken.
In Teil B werden die Probanden nach einer Selbsteinschätzung ihrer Kenntnisse in verschiedenen Bereichen gefragt. Hierbei können sie je Bereich eine der Antworten *Sehr viel*, *Viel*, *Moderat*, *Wenig*, *Sehr wenig*, oder *Keine* ankreuzen. Die Probanden werden zu den

Bereichen *ExplorViz*, *Software-Architektur*, *Softwarevisualisierungstools*, *Softwareentwicklung*, *Webentwicklung*, *Design*, *Objektorientierte Programmierung*, und *GitLab* gefragt. Auch diese Frage ermöglicht uns mögliche Zusammenhänge zwischen den Probanden und der Bewertung der Funktionen aufzudecken.

Wie zuvor erwähnt bestehen die Teile C, D, E, und F jeweils aus Aufgaben und Fragen. In jedem der Teile wird zunächst gefragt, ob die Aufgabe ohne Hilfestellung bearbeitet wurde. Dies ermöglicht uns zu erkennen, ob es bei einer Funktion Probleme in der Benutzung gab. Außerdem gibt es in jedem der Teile geschlossene Fragen, bei denen die Probanden in einer Fünf-Werte-Skala einen Wert ankreuzen sollen. Hierbei stellt die Antwort Eins jeweils die beste und Fünf die schlechteste Antwort dar. Neben den geschlossenen Fragen beinhalten die Teile D, E, und F jeweils offene Fragen, in denen nach allgemeinen Verbesserungsvorschlägen und sonstigen Anmerkungen gefragt wird. Während die Beantwortung der geschlossenen Fragen der Teile C, E, und F wenig Relevanz für diese Arbeit hat, sind die Antworten der offenen Fragen der Evaluation durchaus relevant, da diese einen Einblick über relevante Anmerkungen für die Persistierung und die Schnittstelle liefern können.

Tabelle 6.3. Geschlossene Fragen aus Teil D des Online-Fragebogens [Wulff und Brehme 2024].

ID	Frage
D2	Wie zugänglich ist das Öffnen einer Landscape?
D3	Wie intuitiv fanden Sie das Erstellen einer freien Annotation?
D4	Wie intuitiv fanden Sie das Erstellen einer zugehörigen Annotation?
D5	Wie gut konnten Sie Annotationen bearbeiten?
D6	Wie intuitiv empfanden Sie die Minimierungsfunktion der Annotationen?
D7	Wie einfach war es eine minimierte Annotation wieder zu öffnen?
D8	Wie gut konnten Sie Annotationen löschen?
D9	Wie haben Sie das Teilen von Annotationen in einer kollaborativen Sitzung empfunden?
D10	Wie verständlich ist das Design der Annotationen?
D11	Wie sinnvoll empfinden Sie die Funktion der Annotationen?

In Teil D des Fragebogens werden Fragen zu der Funktion der Annotationen gestellt. Wie zuvor erwähnt wird diese zunächst mit der Frage, ob das Anlegen der Annotationen ohne Hilfestellung funktioniert hat, gestartet. Anschließend werden die in Tabelle 6.3 aufgeführten geschlossenen Fragen gestellt. Die Frage D2 bietet dabei keine Relevanz für die Annotation, gibt aber eine weitere Möglichkeit, die gegebenen Antworten in einem Zusammenhang zu stellen, da das Öffnen einer Landscape eine der Grundfunktionalitäten von ExplorViz ist. Die Fragen D3 bis D8 ermöglichen ein Feedback über die grundsätzlichen Funktionen der Annotationen, während die Frage D9 ein Feedback zu dem Umgang mit Annotationen in Kollaborationsräumen liefert. Für ein Feedback für die Verständlichkeit des Designs der Annotationsfenster stellen wir Frage D10. Die Beantwortung von D11 gibt

eine allgemeine Rückmeldung über die Sinnhaftigkeit der Funktion. Als Letztes bilden, wie zuvor erwähnt, zwei offene Fragen für generelles Feedback den Schluss des Teils D.

6.4. Ergebnisse

In diesem Abschnitt präsentieren wir die Ergebnisse der von uns durchgeführten Usability-Studie. Da die Antworten auf die offenen Fragen und allgemeine Rückmeldungen sehr vielfältig ausfallen, präsentieren wir nur die Ergebnisse der geschlossenen Fragen. Wir werden jedoch auf die Antworten der offenen Fragen im darauffolgenden Abschnitt eingehen. Wie zuvor erwähnt, präsentieren wir aufgrund der Relevanz nur die Ergebnisse für die personenbezogenen Daten und den Teil D für die Annotationen und gehen im Abschnitt 6.5 auch auf die für die Persistierung relevanten Rückmeldungen zu den anderen Teilen ein. Der komplette Datensatz der Ergebnisse ist mitsamt des Fragebogens öffentlich zugänglich [Wulff und Brehme 2024].

Probanden
Es haben insgesamt 14 Personen an der Usability-Studie teilgenommen, von denen 13 Studenten sind und eine Person wissenschaftlicher Mitarbeiter. Von den 13 Studenten studieren drei außerhalb des Fachbereichs der Informatik. Zudem gaben 8 der 14 Probanden an, die allgemeine Hochschulreife, vier einen Bachelorabschluss, und zwei einen Masterabschluss als höchsten Bildungsabschluss zu besitzen.
Wie in Tabelle 6.4 zu sehen, gaben acht der Probanden an, keine Erfahrung in ExplorViz und fünf keine Erfahrung mit Softwarevisualisierungstools zu haben. In den anderen Bereichen gaben maximal drei Personen an, keine Erfahrung in diesen zu haben. Ansonsten waren die Erfahrungen überwiegend, mit Ausnahme des Bereichs Design, im mittleren Bereich zwischen 'Viel' und 'Wenig' verteilt.

Tabelle 6.4. Anzahl je Antwortmöglichkeit für die Bereiche auf die Frage der Erfahrung in den verschiedenen Bereichen [Wulff und Brehme 2024].

Bereich	Sehr viel	Viel	Moderat	Wenig	Sehr wenig	Keine
ExplorViz	0	3	2	1	0	8
Software-Architektur	0	3	5	1	2	3
Softwarevisualisierungstools	0	0	3	3	3	5
Softwareentwicklung	1	3	4	3	0	3
Webentwicklung	1	3	3	2	3	2
Design	0	0	1	5	5	3
Objektorientierte Programmierung	1	7	2	1	1	2
GitLab	0	5	5	1	0	3

Durchführung der Aufgaben
Von den 14 Probanden gaben elf an, dass die Erstellung von Annotationen ohne Hilfestellung funktioniert hat und drei, dass diese Hilfe benötigten.

Benutzbarkeit
Für die Fragen D2 bis D11 hatten die Probanden die Möglichkeit, einen der Werte zwischen eins und fünf zu wählen, wobei eins der bestmöglichen und fünf der schlechtmöglichsten Antwort entspricht. Somit zeigt ein Mittelwert von 2,5 den Ausgleich einer sehr schlechten und einer sehr guten Antwort. Tabelle 6.5 zeigt die zusammengefassten Ergebnisse der geschlossenen Fragen für den Teil D des Fragebogens. Neben dem Mittelwert zur Bewertung der Ergebnisse haben wir außerdem die Standardabweichung als Maß für die Verteilung der Antworten.

Tabelle 6.5. Ergebnisse der geschlossene Fragen für die Annotationsfunktion aus Teil D des Online-Fragebogens in Form von Mittelwert und Standardabweichung je Frage [Wulff und Brehme 2024].

ID	Mittelwert	Standardabweichung
D2	1,21	0,58
D3	1,64	0,93
D4	2,36	1,34
D5	1,36	0,63
D6	1,29	0,83
D7	2,43	1,16
D8	1,14	0,53
D9	1,29	0,61
D10	1,29	0,47
D11	1,36	0,63

Allgemeine Rückmeldung
Auf die Frage D12 '*Was könnte man verbessern?*' antworteten 12 der 14 Probanden und gaben schriftliches Feedback. Zusätzlich schrieben fünf Probanden eine Rückmeldung bei Frage D13 '*Sonstige Anmerkungen*'.

6.5. Diskussion

In diesem Abschnitt diskutieren wir die zuvor genannten Ergebnisse. Die Diskussion unterteilen wir in die Bereiche Erstellen & Löschen, Bearbeiten, Minimieren, Kollaboration, allgemeine Rückmeldung, und Rückmeldungen zur Persistierung.

Erstellen & Löschen
Von den 14 Probanden gaben elf in Frage D1 an, dass die Erstellung der Annotationen ohne Hilfestellung funktioniert hat. Von den drei Probanden, die Hilfe benötigten, hatten zwei keine und der Dritte nur wenig Erfahrung in ExplorViz. Das zeigt, dass die Erfahrung mit ExplorViz hilfreich für die Bearbeitung der Aufgaben war. Die Fragen D3 und D4 stellen hierzu die Frage der intuitiven Erstellung freier und zugehöriger Annotationen. Mit einem Mittelwert von 1,64 für die freien Annotationen in Frage D3 wurde die Erstellung der freien Annotationen äußerst gut bewertet. Frage D4 zeigt allerdings mit einem Mittelwert von 2,36, dass bei der Erstellung der zugehörigen Annotationen noch Verbesserungspotenzial ist. Dies spiegelt sich ebenfalls in den Rückmeldungen zu den Fragen D12 und D13 wider, in der fünf Probanden mitgeteilt haben, dass das Öffnen der zugehörigen Annotation durch Drücken der Umschalt-Taste zusammen mit dem Ziehen mit der Maus kontraintuitiv sei, beziehungsweise einen Verbesserungsvorschlag hierfür gemacht haben. Zudem war dies in den meisten Fällen der Grund, weshalb bei dem Aufgabenteil D nach Hilfe gefragt wurde. Als ein Verbesserungsvorschlag wurde hier genannt, statt des Ziehens mit der Maus eine Schaltfläche zur Erstellung zu nutzen. Wir betrachten diesen Vorschlag als eine sehr gute Verbesserung für eine intuitivere Benutzung. Frage D8 zeigt mit einem Mittelwert von 1,14, dass das Löschen der Annotationen sehr intuitiv ist. Lediglich eine Person schätzte dies mit einer drei ein. Hierzu gab es die Rückmeldung, dass sich das Mülleimer-Icon weiß färbt, wodurch die Sichtbarkeit des Icons eingeschränkt wird. Dies für die Sichtbarkeit zu ändern klingt für uns nach einer logischen Möglichkeit.

Bearbeiten
In Frage D5 werden die Probanden gefragt, wie gut diese die Annotationen bearbeiten konnten. Basierend auf den Ergebnissen, mit einem Mittelwert von 1,36, wurde die Bearbeitung als sehr gut empfunden. Allerdings gab es hier die Rückmeldung, dass die Schaltfläche zum Wechseln in und aus dem Editiermodus leicht übersehen werden kann und es besser sein könnte, diese innerhalb des Eingabefeldes mit einzuarbeiten. Wir empfinden diesen Vorschlag als sinnvoll. Allerdings wird die Schaltfläche für die Synchronisierung der Annotationen mit anderen Kollaborationsteilnehmern genutzt und die allgemeine Rückmeldung zum Bearbeiten ist sehr positiv ausgefallen.

Minimieren
Die Fragen D6 und D7 behandelten die Funktion des Minimierens der Annotationen. Hierbei zielte D6 auf die intuitive Benutzung der Minimierungsschaltfläche ab, während D7 nach der Einfachheit des Öffnens minimierter Annotationen fragte. Die Ergebnisse zeigen, dass das Minimieren selbst, mit einem Mittelwert von 1,29, als sehr intuitiv wahrgenommen wurde. Das erneute Öffnen hingegen wurde als schwieriger empfunden. Der Mittelwert von 2,43 ist zwar im mittleren Bereich, allerdings gab es sechs Anmerkungen in D12 und D13, die eine bessere Sichtbarkeit der zugehörigen Annotationen als Verbesserung vorschlugen. Einer dieser Vorschläge war das Einfärben eines zugehörigen Objekts. Dies

ist eine sehr gute Idee, gegen die wir uns allerdings entschieden haben aufgrund der Highlight-Funktion von ExplorViz.

Kollaboration
Mit einer durchschnittlichen Bewertung von 1,29 wurde das Teilen von Annotationen in einer kollaborativen Sitzung (Frage D9) als sehr gut empfunden. Zudem gab es keine Verbesserungsvorschläge zu der Funktion. Es wurden lediglich kleinere Bugs gemeldet, die bereits behoben wurden.

Allgemeine Rückmeldung
Das Öffnen der Landscape, nach dem in Frage D2 gefragt wird, wurde allgemein mit einem Mittelwert von 1,21 als sehr zugänglich wahrgenommen. Dies zeigt, dass die Probanden keine weiteren Hindernisse mit dem Zugang zur Testumgebung hatten.
In Frage D10 wird nach der Verständlichkeit des Designs der Annotationen gefragt. Diese wurde mit einer durchschnittlichen Bewertung von 1,29 als sehr verständlich empfunden und es gab, neben zuvor angesprochenen Schaltflächen, keine weiteren Rückmeldungen dazu.
Die Sinnhaftigkeit der Annotationsfunktion, die in Frage D11 behandelt wird, wurde mit einem Mittelwert von 1,36 als durchaus sehr sinnvoll eingeschätzt. Hierzu gab es keine weiteren Anmerkungen.
Neben den zuvor erwähnten Anmerkungen gab es noch weitere in den Fragen D12 und D13. Hier wurde unter anderem eine Auflistung der vorhandenen Annotationen empfohlen, da die Fenster sonst viel Fläche in der Ansicht einnehmen. Diesen Vorschlag empfinden wir als sehr sinnvoll, da dies mehr Struktur in die Anwendung bringt. Eine weitere Anmerkung galt der Beschränkung auf eine Annotation pro Objekt. Mit der zuvor erwähnten Auflistung zusammen sehen wir es als unproblematisch an, dass es mehrere Annotationen pro Objekt gibt. In unserem umgesetzten Ansatz haben wir dies jedoch beschränkt, damit die Anzahl an Fenstern in der Ansicht nicht zu groß wird.

Rückmeldungen zur Persistierung
Während die geschlossenen Fragen von Teil C, E, und F lediglich die Benutzung der Oberfläche für die Snapshots und die Git-Hosting-Anbindung thematisieren und somit für diese Arbeit nicht relevant sind, betrachten wir die offenen Fragen dieser Teile, da es zu relevanten Rückmeldungen für die Persistierung dieser kommen kann.
In der Frage E10 konnten die Probanden Verbesserungsvorschläge zu der Snapshotfunktion geben. In dieser wurde als Verbesserungsmöglichkeit genannt, dass aktuell gleiche Snapshotnamen möglich seien und dies unterbunden werden sollte. Während wir die Identifikation der Snapshots über den Ersteller und das Erstellungsdatum machen, da wir die Mehrfachnutzung eines Namens für Snapshots nicht als problematisch sehen, können wir den Vorschlag durchaus verstehen. Dies könnte zu einer besseren Übersicht über vorhandene Snapshots führen.

Fazit
Insgesamt können wir zusammenfassen, dass die Funktionen als sinnvoll erachtet wurden und ihre Benutzbarkeit gut ist. Jedoch sollten die zuvor genannten Rückmeldungen und Verbesserungen umgesetzt werden, damit die Benutzbarkeit der Funktionen einfacher und intuitiver wird.

6.6. Validität der Evaluation

In diesem Abschnitt erläutern wir Bedrohungen für die Validität der Evaluationsergebnisse.

Probanden
Eine Bedrohung für die Validität der Ergebnisse besteht in der Anzahl der 14 Probanden, die an der Evaluation teilgenommen haben, da es schwer ist, aus dieser geringen Anzahl eine Allgemeingültigkeit der Ergebnisse zu erlangen. Außerdem besteht eine Bedrohung bezüglich der Erfahrung in ExplorViz. Acht der 14 Probanden gaben an, keine Erfahrung in ExplorViz zu haben. Wir gehen zwar davon aus, dass Erfahrungen mit ExplorViz für die Evaluation zweitrangig waren, wir können allerdings nicht ausschließen, dass eine Evaluation mit Experten zu anderen Ergebnissen führen könnte. Zudem gaben drei der Probanden an, außerhalb der Informatik beschäftigt zu sein. Auch in diesem Fall gehen wir davon aus, dass dies keine Auswirkungen auf die Testbarkeit der in dieser Arbeit behandelten Funktionen hat. Wir können dies allerdings nicht ausschließen. Außerdem kennen wir jeden der Probanden persönlich, da wir Freunde, Studenten, und Mitarbeiter des Instituts für Informatik der CAU Kiel als Teilnehmer der Evaluationen gewinnen konnten. Dies kann die Ergebnisse positiv verzerrt haben, obwohl wir nach ehrlichem Feedback gefragt haben.

Relevanz der Aufgaben
Eine weitere Bedrohung für die Validität der Ergebnisse stellt die Relevanz der Aufgaben dar. Die Aufgaben der Evaluation zielten darauf ab, die implementierten Funktionen zu testen. Hierfür wurden die Aufgaben sehr simpel gehalten und kein realistischer Kontext für die Nutzung gegeben, der den Einsatz erklärt. Wir haben uns dafür entschieden, da wir den Fokus auf die allgemeine Benutzbarkeit der Funktionen gesetzt haben und die Evaluation in einem zeitlich angemessenen Rahmen halten wollten. Wir gehen davon aus, dass eine reale Nutzung von ExplorViz deutlich komplexer ausfällt.

Variabilität der Einführung in ExplorViz
Die Variabilität der Einführung zu ExplorViz stellt eine weitere Bedrohung für die Validität der Ergebnisse dar. Da wir jeden Probanden vor der Einführung nach seiner Selbsteinschätzung der Kenntnisse zu ExplorViz gefragt haben und dementsprechend die Einführung inhaltlich gestaltet haben, können wir nicht ausschließen, dass hierdurch unterschiedliche

Kenntnisstände zu ExplorViz entstanden sind. Dies kann zum einen daran liegen, dass die Probanden ihre ExplorViz-Kenntnisse fehlerhaft eingeschätzt haben, zum anderen, dass durch die Anpassung der Einführung nicht immer alle benötigten Informationen von uns weitergegeben wurden. Eine allgemeine Einführung, die unabhängig des Kenntnisstands der Probanden gegeben wird, könnte somit zu anderen Ergebnissen führen.

Kapitel 7

Verwandte Arbeiten

In diesem Abschnitt betrachten wir verwandte Arbeiten zum Thema der Annotationen in Softwarevisualisierungen.

7.1. Immersive Software Archaeology

Immersive Software Archaeology (ISA) [Hoff u. a. 2024] ist eine Softwarevisualisierung für Systeme, die in objektorientierten Programmiersprachen entwickelt wurden und dessen Fokus auf dem Verständnis von Software liegt. Anders als ExplorViz ist ISA eine eigenständige VR-Anwendung, deren Backend in die Eclipse IDE integriert wurde, welche lediglich eine statische Analyse von Softwaresystemen unterstützt.
ISA verwendet eine Darstellung von Sphären und Zylindern. In diesem werden das Projekt und die Pakete als verschachtelte Sphären dargestellt, während Klassen als Zylinder dargestellt werden. Dies ist ähnlich dem Prinzip der City Metapher, die in ExplorViz genutzt wird. Neben den Klassen werden allerdings auch die Methoden und Attribute dargestellt. Methoden werden ebenfalls als Zylinder unter- und überhalb des Klassenzylinders dargestellt. Attribute hingegen werden als Spitzen, die seitlich aus dem Klassenzylinder ragen, visualisiert. Ähnlich zu ExplorViz gibt es auch in ISA Linien zwischen den Klassen. Während diese in ExplorViz den dynamischen Nachrichtenaustausch zwischen Klassen darstellen, wird in ISA über diese die Referenz zwischen Klassen dargestellt. Dabei wird unterschieden, ob es ich um eingehende oder ausgehende Referenzen eines ausgewählten Elements handelt. Außerdem wird unterschieden, ob es sich um eine Typreferenz, einen Methodenaufruf, oder einen Zugriff auf ein Attribut handelt. ISA bietet für diese Kategorien Filtermöglichkeiten an. Für eine gute Unterscheidung werden die verschiedenen Sphären der Pakete in ISA durch unterschiedliche Farben dargestellt. Dies ist anders als in ExplorViz, in dem Pakete immer die gleiche Farbe, die vom Benutzer eingestellt werden kann, haben.
Ähnlich zu den in dieser Arbeit thematisierten Annotationen verfügt ISA über die Funktion Whiteboards in der Umgebung zu erstellen. Diese sind frei bewegbar und können durch die VR-Bedienung von Hand beschrieben werden. Außerdem werden die Whiteboards zwischen allen Benutzern synchronisiert. Anders als die Annotationen in ExplorViz, die bisher nicht in den VR-Modus von ExplorViz integriert sind, wurden die Whiteboards in ISA für die Nutzung in einer VR-Umgebung entwickelt. Wie in Abbildung 7.1 gezeigt,

ist es außerdem möglich, in ISA Screenshots zu machen und diese an ein Whiteboard anzupinnen. Ein Screenshot kann, ähnlich zu den Snapshots aus ExplorViz, dazu genutzt werden, eine festgehaltene Ansicht wiederherzustellen. ISA verfügt außerdem über die Möglichkeit, Audioaufnahmen zu machen, die ebenfalls an ein Whiteboard gepinnt werden können. Die Whiteboards können anschließend außerhalb der VR-Umgebung in der IDE betrachtet werden.

Abbildung 7.1. Darstellung von ISA [Hoff u. a. 2024]. Benutzeransicht in ISA mit einem Whiteboard, auf dem Screenshots angepinnt sind.

7.2. FlyThruCode

FlyThruCode (FTC) [Oberhauser u. a. 2016] ist eine 3D-Softwarevisualisierung, die in der Unity Game Engine[1] entwickelt wurde. FTC bietet eine statische Analyse von Softwaresystemen, in welcher diese visuell mithilfe unterschiedlicher Metaphern dargestellt werden. Anders als die von ExplorViz genutzte City Metapher, stellt FTC die Systeme mittels der Universumsmetapher oder der terrestrischen Metapher dar, in welcher der Benutzer zur Navigation die Kamera umherfliegen lassen kann. In der Universumsmetapher werden Pakete als Sonnensysteme, Klassen als Planeten, und Abhängigkeiten zwischen Klassen und Paketen als gerichtete, farbige Lichtstrahlen dargestellt. Zusätzlich wird über die Größe der Planeten die Anzahl an Methoden der jeweiligen Klasse dargestellt. Bei der terrestrischen Metapher werden Pakete als Städte in einer gelabelten Glaskuppel dargestellt. In dieser befinden sich Gebäude, welche die Klassen darstellen und Rohre zwischen den Objekten stellen die Abhängigkeiten dar. In dieser Metapher steht die Anzahl der

[1] https://unity.com/de

Stockwerke der einzelnen Gebäude für die Anzahl der Methoden einer Klasse. Diese verschachtelten Varianten der Darstellungen sind ähnlich der Darstellung in ExplorViz. Neben der dreidimensionalen Darstellung von Softwaresystemen bietet FTC zusätzlich die Möglichkeit betreffenden Quellcode anzuzeigen, die Ansicht zu filtern, und dynamisch UML-Diagramme zu erstellen. Ähnlich zu den Pop-ups aus ExplorViz ist es außerdem möglich sich Metriken zu einzelnen Objekten anzeigen zu lassen.
Vergleichbar mit den in dieser Arbeit behandelten Annotationen verfügt auch FTC über Annotationen. In FTC gibt es zum einen automatisch generierte und platzierte Tags, die an Objekten platziert werden und zum anderen benutzerdefinierte Annotationen. Anders als in ExplorViz kann für diese, wie in Abbildung 7.2 zu sehen ist, ein vorhandener Tag benutzt oder ein neuer erstellt werden sowie zusätzlich eine Markierung und eine Farbe gewählt werden. Wie auch in ExplorViz bietet die Annotation in FTC die Möglichkeit, eine Notiz mittels eines Textfeldes zu schreiben.

FTC verfügt, wie auch ExplorViz, über eine VR-Ansicht. Virtual Reality FlyThruCode (VR-FTC) [Oberhauser und Lecon 2017] ermöglicht die Funktionalitäten von FTC in einer VR-Darstellung. Dabei werden Funktionen, wie die Quellcodeansicht oder die Metriken, über ein virtuelles Tablet dargestellt. Außerdem gibt es die Möglichkeit für Benutzer bereits vorhandene oder selbsterstellte Labels an Objekten zu platzieren, um diese einfacher wiederzufinden.

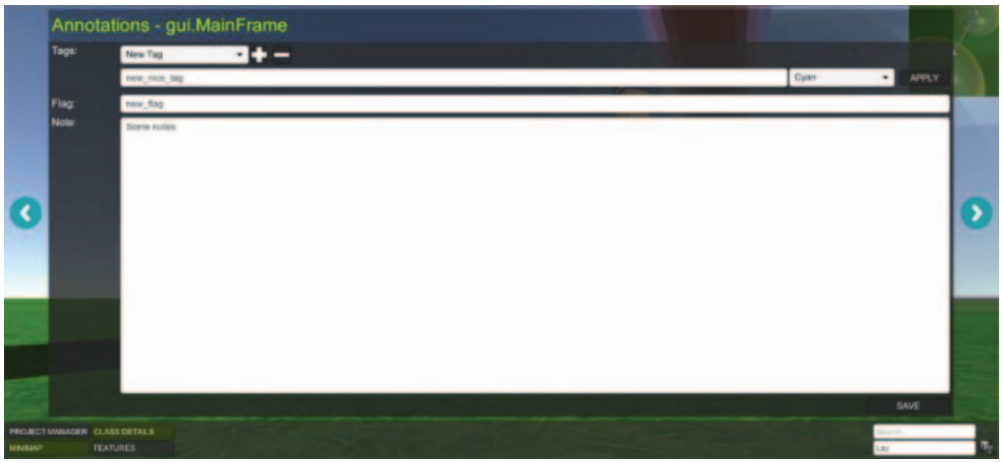

Abbildung 7.2. Annotationserstellung in FTC [Oberhauser u. a. 2016]. Fenster zur Erstellung einer Annotation an einer Klasse in FTC.

Kapitel 8

Fazit und Ausblick

In diesem Kapitel ziehen wir ein Fazit zu der vorgestellten Arbeit und geben einen Ausblick auf zukünftige Arbeiten.

8.1. Fazit

Im Rahmen dieser Bachelorarbeit haben wir einen Ansatz für eine Annotationsfunktion für die 3D-Softwarevisualisierung ExplorViz entwickelt. Die Annotationen können benutzt werden, um kurze oder ausführliche Informationen in der Ansicht festzuhalten. Außerdem sind sie frei in der Ansicht oder einem Objekt zugeordnet. Für die Übersichtlichkeit gibt es die Möglichkeit, die Annotationen zu minimieren und nicht mehr benötigte Annotationen können gelöscht werden. Zudem sind diese in den Kollaborationsmodus von ExplorViz eingebaut. Es ist also möglich, Annotationen mit anderen zu teilen und für alle Teilnehmer zu bearbeiten. Außerdem haben wir einen Ansatz für die Persistierung der Daten für eine Snapshotfunktion und einer Git-Hosting-Anbindung entwickelt. Dieser Ansatz beinhaltet neben der Speicherung der Daten in einer Mongo-Datenbank, ebenfalls eine REST-Schnittstelle, über die mit dem Frontend kommuniziert werden kann.
In einer Usability-Studie haben wir die Benutzbarkeit der implementierten Funktionen mit der Hilfe von 14 Probanden evaluiert. Die Probanden haben mit einem Online-Fragebogen gearbeitet, in dem ihnen Aufgaben zu den verschiedenen Funktionen gestellt wurden. Anschließend sollten die Probanden Fragen zur Benutzung und dem Design der Funktionen beantworten. Die Aufgaben waren so gestellt, dass die Funktionen ausführlich benutzt werden mussten. Die Ergebnisse der Studie haben gezeigt, dass die entwickelten Funktionen sinnvoll für ExplorViz sind und gaben Vorschläge für die weitere Entwicklung.

8.2. Ausblick

In diesem Abschnitt diskutieren wir Ideen für die zukünftige Entwicklung, die aus den Ergebnissen der Evaluation und eigenen Erkenntnissen stammen.

Einbindung in die erweiterte Realität
Da ExplorViz nicht nur über die Darstellung im Browser, sondern auch über die Darstellung

in VR und AR verfügt, sollten die Annotationen in der Zukunft für diese Darstellungsvarianten implementiert werden. Hierfür wäre eine Möglichkeit, ähnlich zu ISA [Hoff u. a. 2024], die Annotationen in der erweiterten Realität frei von Hand beschreibbar zu machen, um die Eingabebarriere über eine Tastatur zu verhindern.

Erstellen von Kollaborationsräumen
Wie bereits erwähnt werden aktuell bei der Erstellung von Kollaborationsräumen die Annotationen, sowie auch die Pop-ups, die zuvor geöffnet wurden, geschlossen. Für die zukünftige Entwicklung sollte es hier ermöglicht werden, dass bei Erstellung des Raumes bereits vorhandene Annotationen erhalten bleiben.

Überarbeitung der Erstellung zugehöriger Annotationen
Die Evaluationsergebnisse lassen darauf schließen, dass das Erstellen von zugehörigen Annotationen als kontraintuitiv wahrgenommen wurde. Hier sollte in Zukunft eine andere Art umgesetzt werden, diese zu erstellen. Eine Idee wäre, dass weiterhin per Hover über ein Objekt das Fenster in der Ansicht erscheint und dort für einige Sekunden offen bleibt. Läuft die Zeit ab, so schließt es sich, sofern es nicht über einen Klick auf eine Schaltfläche endgültig erstellt wurde.

Hervorheben von zugehörigen Objekten
In der Evaluation wurde mehrfach erwähnt, dass es schwer ist ein Objekt einer Annotation zuzuordnen. Hierfür könnte in Zukunft eine Hervorhebung durch das Färben des Randes, ähnlich eines Rahmens, des jeweiligen Objekts implementiert werden. Dies wäre ein stärkerer visueller Hinweis als die aktuelle, vorhandene Labelerweiterung und würde nicht mit der Highlightfunktion von ExplorViz kollidieren.

Übersichtsmenü für die Annotationen
Zur Wahrung der Übersichtlichkeit verfügen die Annotationen bereits über die Möglichkeit minimiert zu werden. Minimierte zugehörige Annotationen können dann durch erneutes Hovern des zugehörigen Objektes wieder geöffnet werden. Um dies für den Benutzer zugänglicher zu machen, könnte in Zukunft ein Übersichtsmenü implementiert werden, in dem alle existierenden Annotationen angezeigt werden. Über dieses könnte es dann möglich gemacht werden, dass durch Knopfdruck eine Annotation wieder geöffnet wird. Das Übersichtsmenü sollte dann neben den Annotationstiteln auch anzeigen, zu welchen Objekten eine zugehörige Annotation gehört und wer der Besitzer der Annotation ist.

Historie von Änderungen
Neben dem Übersichtmenü könnte in der Zukunft auch eine Historie über Veränderungen an Annotationen gespeichert werden. Diese könnte aus den Annotationen selbst oder über das Übersichtsmenü geöffnet werden und sollte die Möglichkeit bieten, vergangene Zustände der Annotation anzuschauen und eventuell wiederherzustellen.

Literaturverzeichnis

[Atkinson und Buneman 1987] M. P. Atkinson und O. P. Buneman. Types and persistence in database programming languages. *ACM Comput. Surv.* 19.2 (Juni 1987), Seiten 105–170. DOI: 10.1145/62070.45066. (Siehe Seite 5)

[Brehme 2024] J. Brehme. Persistierung und Teilen von Ansichten einer 3D Software Visualisierung. Bachelorarbeit. Christian-Albrechts-Universität zu Kiel, Sep. 2024. (Siehe Seiten 2, 29, 31, 35, 36)

[ExplorViz 2024] ExplorViz. *Conceptual design of our software visualization as a service approach*. 2024. URL: https://explorviz.dev/3-architecture/ (besucht am 13. 07. 2024). (Siehe Seite 10)

[Fittkau u. a. 2015] F. Fittkau, S. Roth und W. Hasselbring. ExplorViz: Visual Runtime Behavior Analysis of Enterprise Application Landscapes. In: *23rd European Conference on Information Systems (ECIS 2015)*. Mai 2015. DOI: https://doi.org/10.18151/7217313. (Siehe Seite 11)

[Hasselbring u. a. 2020] W. Hasselbring, A. Krause und C. Zirkelbach. ExplorViz: Research on software visualization, comprehension and collaboration. *Software Impacts* 6 (2020). DOI: https://doi.org/10.1016/j.simpa.2020.100034. (Siehe Seiten 1, 9)

[Hilbrich und Lehmann 2022] M. Hilbrich und F. Lehmann. Discussing Microservices: Definitions, Pitfalls, and their Relations. In: *2022 IEEE International Conference on Services Computing (SCC)*. 2022, Seiten 39–44. DOI: 10.1109/SCC55611.2022.00019. (Siehe Seite 6)

[Hoff u. a. 2024] A. Hoff, M. Lungu, C. Seidl und M. Lanza. Collaborative Software Exploration with Multimedia Note Taking in Virtual Reality. In: *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*. ICPC '24. Lisbon, Portugal: Association for Computing Machinery, 2024, Seiten 346–357. DOI: 10.1145/3643916.3644427. (Siehe Seiten 47, 48, 52)

[Krause-Glau u. a. 2022] A. Krause-Glau, M. Hansen und W. Hasselbring. Collaborative program comprehension via software visualization in extended reality. *Information and Software Technology* 151 (2022). DOI: https://doi.org/10.1016/j.infsof.2022.107007. (Siehe Seite 11)

[Krause-Glau und Hasselbring 2022] A. Krause-Glau und W. Hasselbring. Scalable Collaborative Software Visualization as a Service: Short Industry and Experience Paper. In: *2022 IEEE International Conference on Cloud Engineering (IC2E)*. 2022, Seiten 182–187. DOI: 10.1109/IC2E55432.2022.00026. (Siehe Seite 9)

[Kul und Sayar 2021] S. Kul und A. Sayar. A Survey of Publish/Subscribe Middleware Systems for Microservice Communication. In: *2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. 2021, Seiten 781–785. DOI: 10.1109/ISMSIT52890.2021.9604746. (Siehe Seite 8)

[Lewis und Fowler 2014] J. Lewis und M. Fowler. *Microservices: a Defintion of this new Architectural Term*. 2014. URL: https://martinfowler.com/articles/microservices.html (besucht am 11. 07. 2024). (Siehe Seite 6)

[Meng u. a. 2018] M. Meng, S. Steinhardt und A. Schubert. Application Programming Interface Documentation: What Do Software Developers Want? *Journal of Technical Writing and Communication* 48.3 (2018), Seiten 295–330. DOI: 10.1177/0047281617721853. (Siehe Seite 5)

[Mishra 2017] A. Mishra. The MVVM Architectural Pattern. In: *iOS Code Testing: Test-Driven Development and Behavior-Driven Development with Swift*. Berkeley, CA: Apress, 2017, Seiten 43–60. DOI: 10.1007/978-1-4842-2689-6_3. (Siehe Seite 8)

[Myers und Stylos 2016] B. A. Myers und J. Stylos. Improving API usability. *Commun. ACM* 59.6 (Mai 2016), Seiten 62–69. DOI: 10.1145/2896587. (Siehe Seite 5)

[Nadareishvili u. a. 2016] I. Nadareishvili, R. Mitra, M. McLarty und M. Amundsen. *Microservice Architecture : Aligning principles, practices, and culture*. Sebastopol: O'Reilly Media, 2016. (Siehe Seite 6)

[Oberhauser und Lecon 2017] R. Oberhauser und C. Lecon. Virtual Reality Flythrough of Program Code Structures. In: *Proceedings of the Virtual Reality International Conference - Laval Virtual 2017*. VRIC '17. Laval, France: Association for Computing Machinery, 2017. DOI: 10.1145/3110292.3110303. (Siehe Seite 49)

[Oberhauser u. a. 2016] R. Oberhauser, C. Silfang und C. Lecon. Code structure visualization using 3D-flythrough. In: *2016 11th International Conference on Computer Science & Education (ICCSE)*. 2016, Seiten 365–370. DOI: 10.1109/ICCSE.2016.7581608. (Siehe Seiten 48, 49)

[Plaisant 2004] C. Plaisant. The challenge of information visualization evaluation. In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI '04. Gallipoli, Italy: Association for Computing Machinery, 2004, Seiten 109–116. DOI: 10.1145/989863.989880. (Siehe Seite 35)

[Robillard 2009] M. P. Robillard. What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software* 26.6 (2009), Seiten 27–34. DOI: 10.1109/MS.2009.193. (Siehe Seite 5)

[Stylos u. a. 2009] J. Stylos, A. Faulring, Z. Yang und B. Myers. Improving API documentation using API usage information. In: Sep. 2009, Seiten 119–126. DOI: 10.1109/VLHCC.2009.5295283. (Siehe Seite 5)

[three.js 2024] three.js. *Fundamentals*. 2024. URL: https://threejs.org/manual/#en/fundamentals (besucht am 27. 08. 2024). (Siehe Seite 9)

[Wettel und Lanza 2007] R. Wettel und M. Lanza. Visualizing Software Systems as Cities. In: *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. 2007, Seiten 92–99. DOI: 10.1109/VISSOF.2007.4290706. (Siehe Seiten 1, 9)

[Wettel u. a. 2011] R. Wettel, M. Lanza und R. Robbes. Software systems as cities: a controlled experiment. In: *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, Seiten 551–560. DOI: 10.1145/1985793.1985868. (Siehe Seite 1)

[Wulff und Brehme 2024] P. Wulff und J. Brehme. *Evaluationsergebnisse - Kollaborative Annotationen und die Persistierung durch Snapshots in einer 3D Software Visualisierung (Bachelorarbeit) & Persistierung und Teilen von Ansichten einer 3D Software Visualisierung (Bachelorarbeit)*. Zenodo, Sep. 2024. DOI: 10.5281/zenodo.13684190. (Siehe Seiten 38–42)

[Xia u. a. 2018] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan und S. Li. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. *IEEE Transactions on Software Engineering* 44.10 (2018), Seiten 951–976. DOI: 10.1109/TSE.2017.2734091. (Siehe Seite 1)

[Zirkelbach u. a. 2018] C. Zirkelbach, A. Krause und W. Hasselbring. On the Modernization of ExplorViz towards a Microservice Architecture. In: *Combined Proceedings of the Workshops of the German Software Engineering Conference 2018*. Band Online Proceedings for Scientific Conferences and Workshops. Ulm, Germany: CEUR Workshop Proceedings, Feb. 2018. URL: http://ceur-ws.org/Vol-2066/. (Siehe Seite 10)