Persistierung und Teilen von Ansichten einer 3D Software Visualisierung

Julius Brehme

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

Moderne Softwaresysteme entwickeln sich kontinuierlich weiter und werden dabei immer komplexer. Folglich werden auch die Wartung und Erweiterung der Software solcher Systeme immer wichtiger und es wird ein erheblicher Teil der Zeit in der Softwareentwicklung für diese Prozesse aufgewendet. Um Entwicklern dabei zu helfen, den Überblick zu behalten, und um neuen Entwicklern eine einfache Einarbeitung in ein bestehendes System zu ermöglichen, gibt es Visualisierungstools, die Softwaresysteme analysieren und darstellen.
In dieser Arbeit wurde das 3D Software Visualisierungstool ExplorViz um eine Persistierung von Ansichten erweitert. Die Ansicht einer Softwarelandschaft bietet eine Übersicht über die Software. In der Ansicht ist eine Veränderung der Kameraperspektive und die Hervorhebung verschiedener Pakete und Klassen möglich. In dieser Arbeit wurde die Möglichkeit, eine bearbeitete Ansicht der Landschaft zu persistieren, implementiert. Zudem ist es möglich, eine Ansicht der Landschaft zu teilen und wieder aufzurufen. Die vorhandene Issueerstellung wurde erweitert, sodass in einem Issue die persistierte Ansicht der Landschaft direkt hinterlegt werden kann. Außerdem wurde der Prozess der Erstellung eines Issues überarbeitet, um eine verbesserte Nutzererfahrung zu bieten.
Die Implementierung wurde im Zuge einer Evaluation von 14 Probanden getestet. Die Probanden hatten unterschiedlich viel Hintergrundwissen und Erfahrung in dem Gebiet der Softwareentwicklung, welches sich auch in den vielseitigen Kommentaren widerspiegelt. Die Evaluation ergab ein vielversprechendes Ergebnis, dass die Implementierung zuverlässig funktioniert und die im Zuge dieser Arbeit entwickelte Erweiterung eine sinnvolle Ergänzung zu ExplorViz darstellt. Des Weiteren wurden durch die ausführliche Rückmeldung in der Evaluation mögliche Anpassungen an den umgesetzten Ansatz aufgezeigt. Wir nutzen das gesamte Feedback, um Ideen zur Erweiterung unseres Ansatzes vorzustellen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Moderne Softwaresysteme entwickeln sich kontinuierlich weiter und werden dabei immer komplexer. Folglich werden auch die Wartung und Erweiterung der Software solcher Systeme immer wichtiger und es wird ein erheblicher Teil der Zeit in der Softwareentwicklung für diese aufgewandt [Xia u. a. 2018]. Um Entwicklern dabei zu helfen, den Überblick zu behalten und neuen Entwicklern eine einfache Einarbeitung in ein bestehendes System zu ermöglichen, gibt es Visualisierungstools, die ein Softwaresystem analysieren und darstellen [Wettel u. a. 2011]. Ein Ansatz in diesem Bereich ist das Konzept der Code Cities, bei dem ein System in einer dreidimensionalen Darstellung visualisiert wird [Wettel und Lanza 2007]. Hierbei repräsentieren Pakete Gebäudeblöcke, während Subpakete und Klassen die einzelnen Gebäude innerhalb dieser Blöcke darstellen. ExplorViz [Hasselbring u. a. 2020] nutzt diesen Ansatz ebenfalls und verwendet Echtzeitdaten aus dynamischen Analysen, um die Kommunikationspfade zwischen den Paketen und Klassen zu zeigen. Eine visualisierte Software wird dabei als eine Landschaft bezeichnet. Dies soll Entwicklern ermöglichen, die Struktur und Interaktionen innerhalb eines Systems besser zu verstehen und analysieren zu können.

1.1. Motivation

Ein häufig auftretendes Szenario ist, dass ein Entwickler in ExplorViz eine Umgebung schafft, in der potenzielle Fehlerquellen gut sichtbar und klar dargestellt sind. Sinnvoll wäre, diese Ansicht mit anderen Entwicklern, die diese Ansicht zu einem beliebigen Zeitpunkt aufrufen können, zu teilen. Bisher ist dies nur über eine Erstellung eines Screenshots möglich, bei dem die anderen Entwickler diese Ansicht selbstständig nachbauen müssen. Alternativ kann die Ansicht über eine Kollaborationssession geteilt werden, mit dem Nachteil, dass hierbei die Entwickler alle zeitgleich zusammenkommen müssen. Daher soll ExplorViz erweitert werden, sodass es möglich ist, eine Ansicht einer Landschaft zu persistieren. Dies ermöglicht Entwicklern, eine erstellte Ansicht der Landschaft und die vom Benutzer gewählte Darstellung längerfristig zu speichern und über einen Abruflink mit anderen Entwicklern zu teilen. Weiterhin wird die Git-Anbindung um die Persistierungsfunktion erweitert, um die Erstellung von Issues mit einer gespeicherten Ansicht der Landschaft zu erleichtern.

1.2. Dokumentenstruktur

Im folgenden Kapitel 2 gehen wir auf die zu erreichenden Ziele ein. Anschließend behandeln wir in Kapitel 3 die nötigen Grundlagen und Technologien, um unsere Ziele zu erreichen. Das Kapitel 4 behandelt den Ansatz für die Umsetzung der Ziele. Kapitel 5 befasst sich mit der Implementierung. Hierbei wird auf das visuelle Design und auf die technische Umsetzung eingegangen. In Kapitel 6 wird die resultierende Implementierung evaluiert. Einen Einblick in verwandte Arbeiten bietet das Kapitel 7. Das letzte Kapitel 8 schließt mit einem Fazit ab und gibt noch einen Ausblick über zukünftige Erweiterungen.

Kapitel 2

Ziele

Das übergeordnete Ziel ist die Persistierung und das Teilen von Ansichten einer Landschaft mit allen dazugehörigen Einstellungen, geöffneten Fenstern und Hervorhebungen zu implementieren. Darüber hinaus soll die Anbindung an einen Git-Hosting Dienst erweitert werden. Um dieses Gesamtziel zu erreichen, wurde eine Unterteilung in spezifische Unterziele vorgenommen, welche die notwendigen Schritte zur Zielerreichung darstellen.

2.1. Z1: Persistierung und Teilen der Landschaft mit deren Ansicht

Das Ziel der Persistierung der Landschaft mit deren Ansicht ist es, Benutzern zu ermöglichen, eine vollständige Landschaft inklusive Einstellungen, geöffneten Fenstern und den Hervorhebungen zu speichern, welcher *Snapshot* genannt wird. Außerdem soll ein *Snapshot* mit anderen Benutzern geteilt werden können. Ein Benutzer soll zudem die Möglichkeit haben, alle *Snapshots* zu verwalten. Des Weiteren soll es möglich sein, *Snapshots* als Dateien zu exportieren und diese auch wieder zu importieren.

2.2. Z2: Erweiterung der Git-Hosting Anbindung

Die Erweiterung der Git-Hosting Anbindung zielt darauf ab, die Integration des Git-Hosting-Dienst GitLab in ExplorViz zu verbessern. Dies umfasst die Möglichkeit, *Snapshots* direkt in Issues zu integrieren und die Benutzerfreundlichkeit bei der Erstellung eines Issues zu verbessern.

2.2.1. Z2.1: Benutzerverwaltung

Die Eingabe des API-Tokens soll refaktoriert werden. Die bisher im *Restructuring*-Menü vorhandene Eingabefläche wird im Benutzerkontext neu implementiert und der benutzerspezifische API-Token soll persistiert werden. Es soll möglich sein, dass ein Benutzer mehrere API-Token anlegen kann und diese eigenständig verwalten kann.

2.2.2. Z2.2: Erweiterung des *Restructuring*-Menüs

ExplorViz bietet die Möglichkeit, ein GitLab-Issue in dem *Restructuring*-Menü zu erstellen. Die Benutzerfreundlichkeit bei der Auswahl des API-Tokens, sowie die Auswahl eines Projektes, in dem das Issue erstellt werden soll, wird überarbeitet. Außerdem soll die Erstellung von Issues erweitert werden, um direkt einen *Snapshot*, wie in 2.1 erläutert, zu erstellen und in dem Issue zu hinterlegen.

2.3. Z3: Evaluation

Um erste Einblicke in das Potenzial und die Grenzen der Entwicklung der neuen Features zu bekommen, soll eine Fallstudie zu der Benutzerfreundlichkeit der neu implementierten Features stattfinden. Das Ziel der Evaluation besteht darin, die Benutzerfreundlichkeit der Features zu testen. Dabei werden Probanden herangezogen, um eine Rückmeldung zur Benutzerfreundlichkeit sowie Verbesserungsvorschläge zu erhalten. Die Probanden sollen aus einer repräsentativen Gruppe von Benutzern bestehen, die unterschiedliche Erfahrungsstufen und Nutzungshäufigkeiten von ExplorViz aufweisen. Die Benutzerfreundlichkeit der neuen Features, insbesondere der Snapshoterstellung und das Teilen eines *Snapshots* sowie die Git-Hosting Anbindung, wird durch gezielte Usability-Tests geprüft. Die Probanden werden verschiedene Szenarien durchlaufen, um die Funktionalität zu testen.

Kapitel 3

Grundlagen und Technologien

3.1. Persistierung

Persistierung ist ein fundamentaler Bestandteil moderner Informationssysteme, der sicherstellt, dass Daten dauerhaft gespeichert und bei Bedarf wieder aufgerufen werden können. Persistierung bezeichnet den Vorgang, Daten über die Laufzeit eines Programms hinaus dauerhaft zu speichern. Für die Persistenz werden bestimmte Prinzipien vorausgesetzt. Die Persistenz ist nicht auf bestimmte Typen beschränkt und es sollen beliebige Eigenschaften persistiert werden können. Zudem sollen alle Werte die gleichen Rechte auf Persistenz haben und ein Wert wird mit seinem Typ gespeichert. Das Speichern eines Wertes mit seinem Typen soll verhindern, dass dieser Wert mit einem anderen als dem zugehörigen abgespeicherten Typ gelesen wird [Atkinson und Buneman 1987].

3.2. Programmierschnittstelle

Eine Programmierschnittstelle, im Englischen Application Programming Interface (API) genannt, bildet die Grundlage für die Kommunikation zwischen Softwarekomponenten. Eine API stellt Dienste oder Daten über eine Reihe von vordefinierten Ressourcen wie Methoden, Objekten oder Uniforme Resource Identifier (URI) bereit. Anwender können durch die Nutzung dieser Ressourcen auf die Daten oder Dienste zugreifen und müssen die zugrunde liegenden Objekte und Verfahren nicht selbst implementieren. APIs sind für viele moderne Softwarearchitekturen zentrale Bestandteile und stellen eine Abstraktion dar, die Programmieraufgaben erleichtern soll und das Wiederverwenden von Code ermöglichen. Außerdem wird durch APIs das Design von verteilten und modularen Softwareanwendungen unterstützt [Meng u. a. 2018].

3.3. Microservice

Microservices sind eine moderne Architektur für die Entwicklung von Softwareanwendungen, bei der eine Anwendung in eine Reihe kleiner, unabhängiger Dienste aufgeteilt wird, die jeweils eine spezifische Anforderung erfüllen [Hilbrich und Lehmann 2022]. Jeder Microservice läuft in seinem eigenen Prozess ab und kommuniziert mit anderen Diensten

über leichtgewichtige Mechanismen, häufig über HTTP-basierte APIs. Diese Architektur ermöglicht eine hohe Flexibilität und Skalierbarkeit. Ein zentrales Merkmal von Microservices ist, dass jeder Dienst unabhängig entwickelt, bereitgestellt und skaliert werden kann und die Anpassung an veränderte Anforderungen erleichtert. Die Unabhängigkeit der Dienste trägt auch positiv zur Fehlertoleranz bei, da das Versagen eines Dienstes nicht zwangsläufig die gesamte Anwendung beeinträchtigt [Nadareishvili u. a. 2016].

3.4. MongoDB

MongoDB[1] ist ein dokumentenorientiertes NoSQL-Datenbankmanagementsystem, welches für seine Flexibilität und Skalierbarkeit bekannt ist. Im Gegensatz zu traditionellen relationalen Datenbanken speichert MongoDB Daten in JSON-ähnlichen Dokumenten, wodurch eine schemalose Struktur ermöglich wird. Durch die Organisation dieser Dokumente in Sammlungen (Collections) wird die Modellierung komplexer, hierarchischer Datenstrukturen vereinfacht.

3.5. Ember.js

Ember.js[2] ist ein JavaScript-Framework, das genutzt wird, um Webapplikationen zu entwickeln. Ember.js basiert auf dem Model-View-ViewModel (MVVM) Muster, welches die Trennung von Logik und Darstellung fördert. Das MVVM-Muster erweitert dabei das MVC-Muster, fasst die Controller- und View-Einheiten zusammen und lagert die gesamte Logik der Darstellung aus dem Controller in ein Viewmodel-Objekt aus. Das Viewmodel-Objekt dient als Kommunikationskanal zwischen dem Model und dem View/Controller [Mishra 2017]. Ein zentrales Merkmal von Ember.js ist die Integration mit dem Ember-CLI Tool, welches das Erstellen, Testen und Bereitstellen von Ember-Anwendungen vereinfacht.

Grundfunktion von Ember.js

Die grundlegenden Bestandteile eine Ember.js Anwendung sind die Router, die Route-Handlers, die Modelle, die Templates, und die Komponenten. Die Abbildung 3.1 verschafft einen Überblick über die Funktionsweise einer Ember Applikation. Der Router verbindet die URLs mit den Route-Handlern, welche die Modelle laden und die Templates rendern. Die Modelle repräsentieren den persistenten Zustand der Anwendung und können Daten zu einem Webserver oder anderen Speicherorten speichern. Die Templates nutzen die Handlebar-Syntax[3], um dynamische Benutzeroberflächen zu erstellen. Die Komponenten sind wiederverwendbare, selbständige UI-Elemente, die sowohl Templates als auch JavaScript-Code enthalten können.

[1] https://mongodb.com/
[2] https://Ember.js.com/
[3] https://handlebarsjs.com/

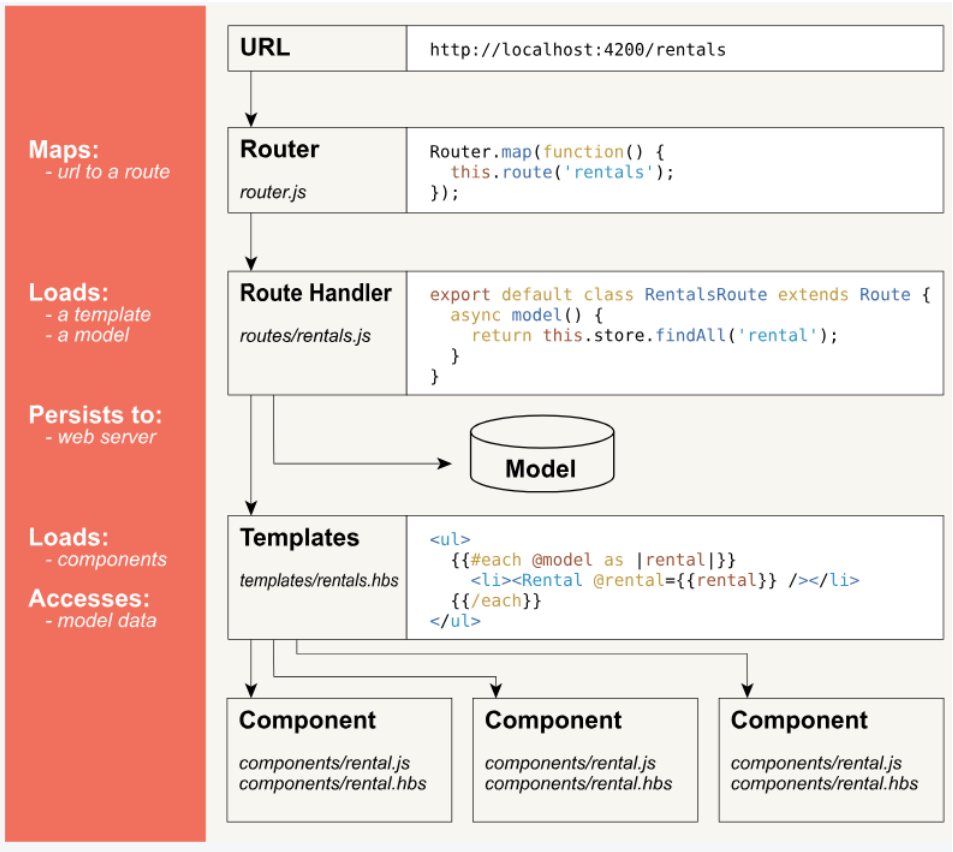


Abbildung 3.1. Die grundlegende Anatomie einer Ember Applikation [Ember.js Autoren 2024].

3.6. three.js

three.js[4] ist eine JavaScript-Bibliothek die es ermöglicht, 3D-Grafiken direkt im Webbrowser darzustellen. Sie baut auf WebGL[5] auf, einer browserinternen API, die es erlaubt, 3D-Grafiken ohne zusätzliche Plugins zu rendern. Mit three.js können Entwickler komplexe 3D-Modelle, Animationen, Interaktionen, und Lichteffekte umsetzen. In three.js spielt die Kamera eine zentrale Rolle, da sie die Blickwinkel bestimmt, aus denen die Szene gesehen wird.

[4]https://threejs.org/
[5]https://www.khronos.org/webgl/

3.7. ExplorViz

ExplorViz [Hasselbring u. a. 2020] ist ein webbasiertes Open-Source 3D-Visualisierungswerkzeug, dass Softwareumgebungen mit Echtzeitdaten dynamischer Analysen visualisiert. Es dient als Hilfestellung zum Erlangen eines besseren Programmverständnis von Softwareumgebungen und Anwendungen. Der Benutzer kann die Softwarelandschaft erkunden und die Code City Metapher wird verwendet, um in der interaktiven Umgebung die Details der Analyse zu zeigen. ExplorViz bietet außerdem die Möglichkeit, die 3D-Umgebung mittels virtueller oder erweiterter Realität zu erkunden.
Als nächstes werden wir uns kurz die Architektur von ExplorViz anschauen sowie auf die Visualisierung einer Softwarelandschaft eingehen.

3.7.1. Architektur

ExplorViz verwendet eine Microservice-Architektur zur Echtzeitanalyse eingehender Daten. In Abbildung 3.2 sind die implementierten Microservices sowie der Datenfluss dargestellt.

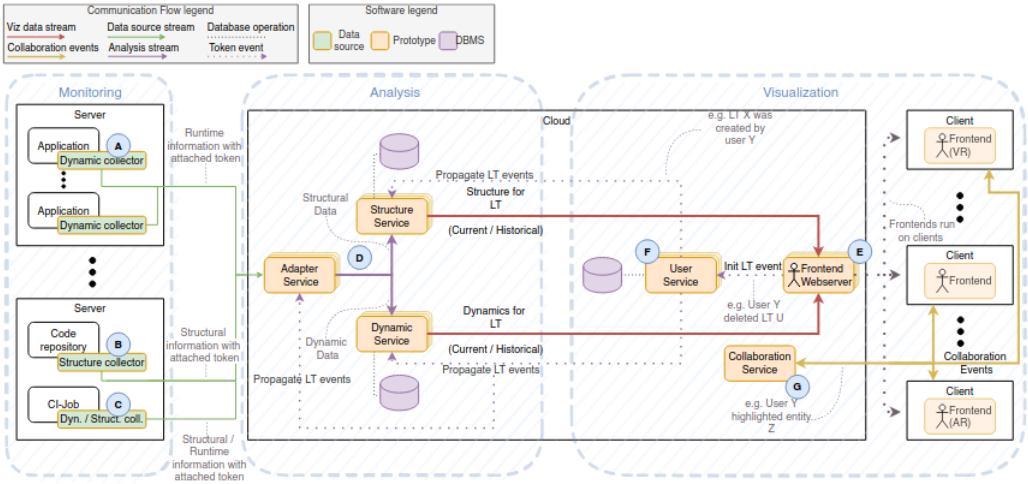


Abbildung 3.2. Übersicht des konzeptuellen Designs der ExplorViz-Architektur [Krause-Glau und Hasselbring 2022].

Wir werden nur auf die Microservices eingehen, die für diese Arbeit relevant sind. Der *Adapter-Service* sammelt und validiert alle Daten aus eingehenden Monitoring-Quellen und extrahiert statische Informationen aus dynamischen Daten. Der *Adapter-Service* verarbeitet anschließend die eingehenden Daten und teilt diese in strukturelle und dynamische Daten auf (Abbildung 3.2 - D). Der *Structure-Service* erstellt eine Baumdarstellung der Anwendungsstruktur auf Basis der statischen und strukturellen Eigenschaften. Der *Dynamic-Service*

führt eine dynamische Analyse durch und erstellt Traces basierend auf den dynamischen Daten. Das Frontend visualisiert die analysierten Daten und ermöglicht Benutzern eine interaktive Erkundung der Softwarelandschaft. Der *User-Service* verwaltet benutzerbezogene Ereignisse und leitet sie an die Analysedienste weiter (Abbildung 3.2 - F). Diese Dienste zusammen ermöglichen eine Analyse und Visualisierung sowohl statischer als auch dynamischer Daten von Softwareanwendungen.

3.7.2. Landschaftsansicht

Abbildung 3.3 zeigt einen Screenshot von der Ansicht einer Landschaft, die im Browser visualisiert wird. Es wird die Städte-Metapher verwendet, um Softwarestrukturen darzu-

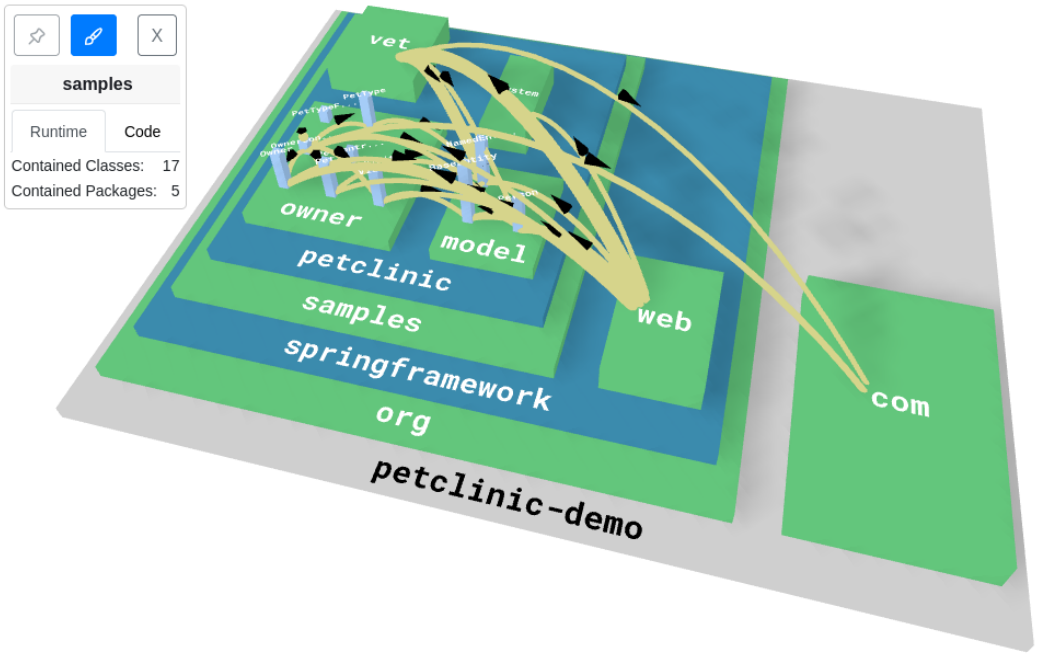


Abbildung 3.3. Ein Screenshot von ExplorViz, welches die PetClinic als Beispielanwendung visualisiert. Ein geöffnetes Pop-Up ist links im Bild zu sehen.

stellen. Pakete erscheinen als grüne Bezirke, die Klassen (blaue Gebäude) oder weitere Pakete enthalten können. Sind Pakete ineinander geschachtelt, werden sie für eine bessere Übersicht in grün und blau dargestellt. Außerdem ist es dem Benutzer möglich, Pakete ein und auf zuklappen. Die Höhe eines Klassenmodells zeigt die Anzahl der Objekte, die zu dieser Klasse gehören. Kommunikation zwischen den Klassen wird durch eine gelbe Linie

dargestellt, wobei die Linienbreite die Anzahl der Methodenaufrufe widerspiegelt. Benutzer können einzelne Klassen und Komponente hervorheben und zusätzliche Informationen in Form von Pop-Ups abrufen [Krause-Glau und Hasselbring 2022].

3.7.3. *Restructuring*

In ExplorViz ist es möglich, Restrukturierung von Softwarecode direkt visuell zu planen und zu dokumentieren. Die *Restructuring*-Komponente beinhaltet einen Modus, der verschiedene Restrukturierungsaktionen anbietet. Erstellen, Umbenennen, Löschen, und Verschieben von Code-Elementen sind bereitgestellt. Um die Funktionen zu nutzen, muss das *Restructuring*-Menü in der Seitenleiste geöffnet werden. Des Weiteren ist es möglich, aus dem *Restructuring*-Menü ein Issue zu erstellen und auf GitLab direkt hochzuladen.

Kapitel 4

Ansatz

In diesem Kapitel wird auf den Ansatz der Persistierung der Ansicht einer Landschaft, einen *Snapshot*, eingegangen. Zuerst werden die generellen Ideen für die Persistierung vorgestellt und die Mockups für das Frontend beschrieben. Danach wird auf den Ansatz über die Erweiterung der Git-Hosting Anbindung mit der Benutzerverwaltung eingegangen. Hierfür wird auf die Integration der Git-Hosting Anbindung eingegangen und welche Anpassungen und Erweiterungen notwendig sind.

4.1. Persistierung der Ansicht von Landschaften

Das Ziel der Persistierung der Ansicht einer Landschaft ist es, die Ansicht bei einem späteren Aufruf exakt und ohne jegliche Veränderungen wiederherstellen zu können. Hierbei sollen sowohl die aktuellen Landschaftsdaten als auch die Kameraposition gespeichert werden. Der Benutzer soll die Möglichkeit haben, entweder die Ansicht der Landschaft zu speichern oder diese als Datei zu exportieren. Außerdem soll es dem Benutzer möglich sein, die gespeicherten Ansichten von Landschaften zu verwalten sowie diese mit anderen Benutzern zu teilen. Das Teilen eines *Snapshots* ist über die Erstellung einer URL möglich. Damit ein geteilter *Snapshot* nicht zeitlich unbegrenzt für andere Benutzer einsehbar ist, soll bei dem Teilen ein Ablaufdatum gesetzt werden können.

4.1.1. Datenstruktur zur Speicherung eines *Snapshots*

Zunächst muss eine geeignete Datenstruktur definiert werden, die alle notwendigen Informationen zu der Landschaft und der Kameraposition enthält. Die Daten werden in einem JSON-Objekt gespeichert und die Struktur ist in Listing 4.1 aufgeführt.

Listing 4.1. Die Datenstruktur für einen *Snapshot* mit der einzelnen, gespeicherten Landschaft und der zugehörigen Ansicht.

```
1 {
2     owner: string,
3     createdAt: number,
4     name: string,
5     landscapeToken: JSON,
6     structureData: {
```

```
7           structureLandscapeData: JSON,
8           dynamicLandscapeData: JSON,
9       },
10      serializedRoom: JSON,
11      timestamps: JSON[],
12      camera: JSON,
13      isShared: boolean,
14      subscribedUsers: {
15          subscriberList: string[]
16      },
17      deleteAt?: number
18  }
```

Das Attribut *owner* hinterlegt den Besitzer eines *Snapshots*. Nur der Besitzer eines *Snapshots* kann diesen auch Löschen. Als *subscribedUsers* werden alle Benutzer festgehalten, die einen geteilten *Snapshot* geöffnet haben. Der Besitzer eines geteilten *Snapshots* kann sich durch das Löschen des *Snapshots* wieder aus der Liste austragen. Die Attribute *landscapeToken* und *structureData* speichern die Informationen, um die Standardansicht einer Landschaft zu laden. Um die gespeicherte Ansicht wiederherzustellen, werden weitere relevanten Daten in dem *serializedRoom* gespeichert, um die genaue Ansicht der Landschaft zu laden. Für die Kameraposition wird ein weiteres Attribut *camera* angelegt, welches die $x-$, $y-$ und z-Koordinaten der Kameraposition speichert. Die *timestamps* werden benötigt, um die Zeitleiste zu rekonstruieren. Das Attribut *deleteAt* ist ein optionales Attribut, in dem das Ablaufdatum als Zeitpunkt eines geteilten *Snapshot* gespeichert wird.

4.1.2. Wiederaufruf der Ansicht

Bei einem Wiederaufruf eines *Snapshots* werden die gespeicherten Daten gelesen und die Datenstruktur wiederhergestellt. Die Landschaft wird dementsprechend der gespeicherten Informationen neu aufgebaut. Dadurch soll der *Snapshot* exakt so wiederhergestellt werden, wie dieser beim Abspeichern zu sehen war. Für die geteilten *Snapshots* sollen anhand der URL die gespeicherten Daten gelesen werden. Damit wird auch der Aufruf eines geteilten *Snapshots* per URL möglich sein.

4.1.3. Mockup der angepassten Startseite

Auf der Startseite sollen alle von dem Benutzer erstellten *Snapshots* sichtbar sein. Dazu zählen auch geteilte *Snapshots*. Die Startseite wird um einen weiteren Abschnitt *Snapshots* erweitert. Dieser ist unterteilt in den Bereich *Personal Snapshots*, *Shared Snapshots* sowie *Subscribed Snapshots*, wie in Abbildung 4.1 zu sehen. Dem Benutzer ist es hier möglich, die *Snapshots* zu verwalten sowie exportierte *Snapshot-Dateien* zu importieren. Unter *Personal Snapshots* werden dem Benutzer alle *Snapshots* angezeigt, die von dem Benutzer selbst

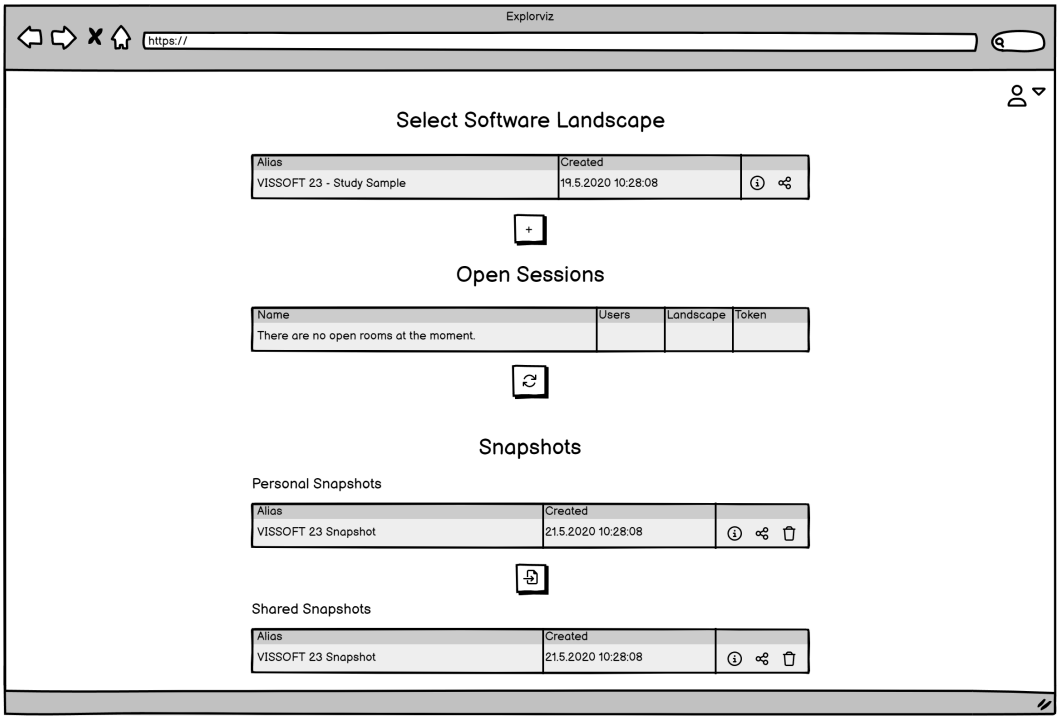


Abbildung 4.1. Mockup für die angepasste Startseite. Die Startseite zeigt persönliche und geteilte *Snapshots* an. Die abonnierten *Snapshots* sind unterhalb der geteilten *Snapshot*. Die Startseite ist scrollbar, um auch die abonnierten *Snapshots* einsehen zu können.

angelegt wurden. Unter *Shared Snapshots* sind die vom Benutzer geteilten *Snapshots* zu finden. Für diese beiden Kategorien hat der Benutzer die Möglichkeit, die *Snapshots* über das Teilen-Icon mit anderen Benutzern zu teilen. Unter *Subscribed Snapshots* werden die von anderen Benutzern geteilten *Snapshots*, die der Benutzer geöffnet hat, angezeigt. Diese Kategorie soll ermöglichen, nicht immer die URL eines geteilten *Snapshots* heraussuchen zu müssen, sondern direkten Zugriff auf diese zu haben. Für alle drei Kategorien hat der Benutzer die Möglichkeit, über das Mülleimer-Icon die *Snapshots* zu löschen sowie über das Informations-Icon Metadaten anzeigen zu lassen.

4.1.4. Mockup des *Snapshot*-Menüs

Um eine Ansicht einer Landschaft zu speichern, wird das vorhandene Menü um den Menüpunkt *Snapshot* erweitert. Wie in Abbildung 4.2 zu sehen, wird dem *Snapshot* ein Name gegeben. Danach ist es möglich, den *Snapshot* entweder zu speichern oder zu exportieren. Anschließend wird der *Snapshot* mit dem jeweiligen Namen auf der Startseite

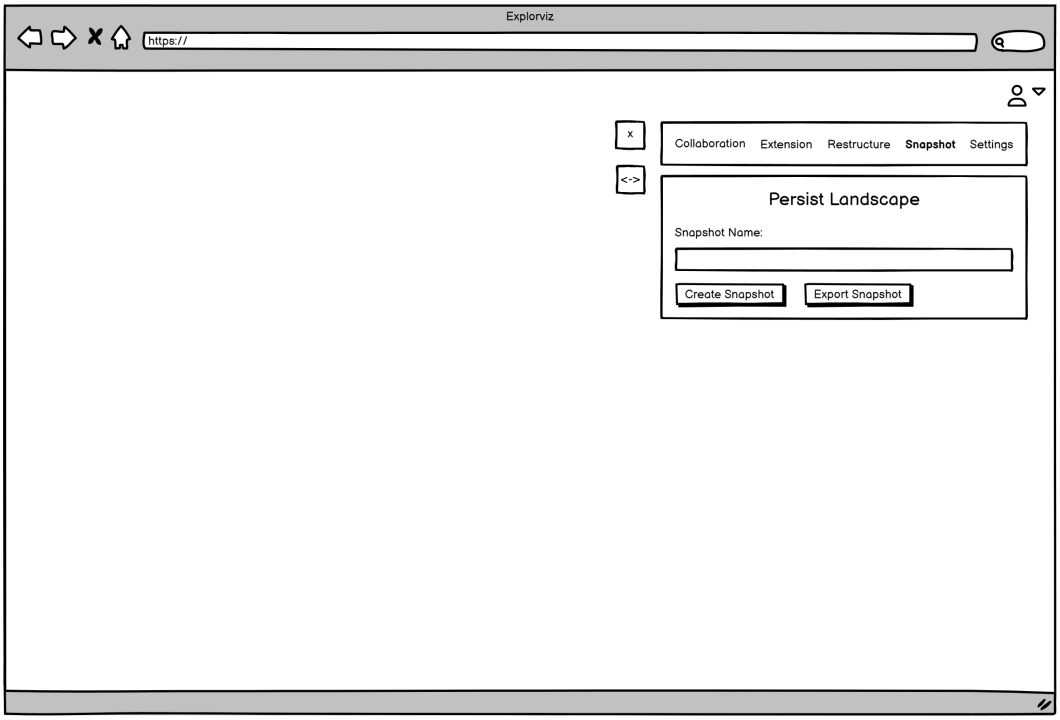


Abbildung 4.2. Mockup für das *Snapshot*-Menü, um die aktuelle Landscape zu speichern.

zur Verfügung gestellt.

4.2. Git-Hosting Anbindung

Ein Benutzer hat in dem *Restructuring*-Menü die Möglichkeit, ein Issue für ein GitLab-Projekt zu erstellen. Die Erstellung eines Issues über das *Restructuring*-Menü ist aber wenig benutzerfreundlich. Im Folgendem wird ein Ansatz vorgestellt, der dem Benutzer die Erstellung eines Issues erleichtern soll, sowie die Erstellung eines geteilten *Snapshots*, welcher direkt in das Issue eingebunden wird, ermöglicht.

4.2.1. Benutzerverwaltung

Um die Git-Hosting Anbindung zu vereinfachen, wird eine Benutzerverwaltung angelegt. Die neu eingerichtete Seite ist in Abbildung 5.4 zu sehen. Auf dieser Seite soll es möglich sein, die verschiedenen Benutzereigenschaften zu verwalten. In der Verwaltung ist es dem Nutzer auch möglich, API-Tokens zu persistieren. Für das Anlegen eines API-Token

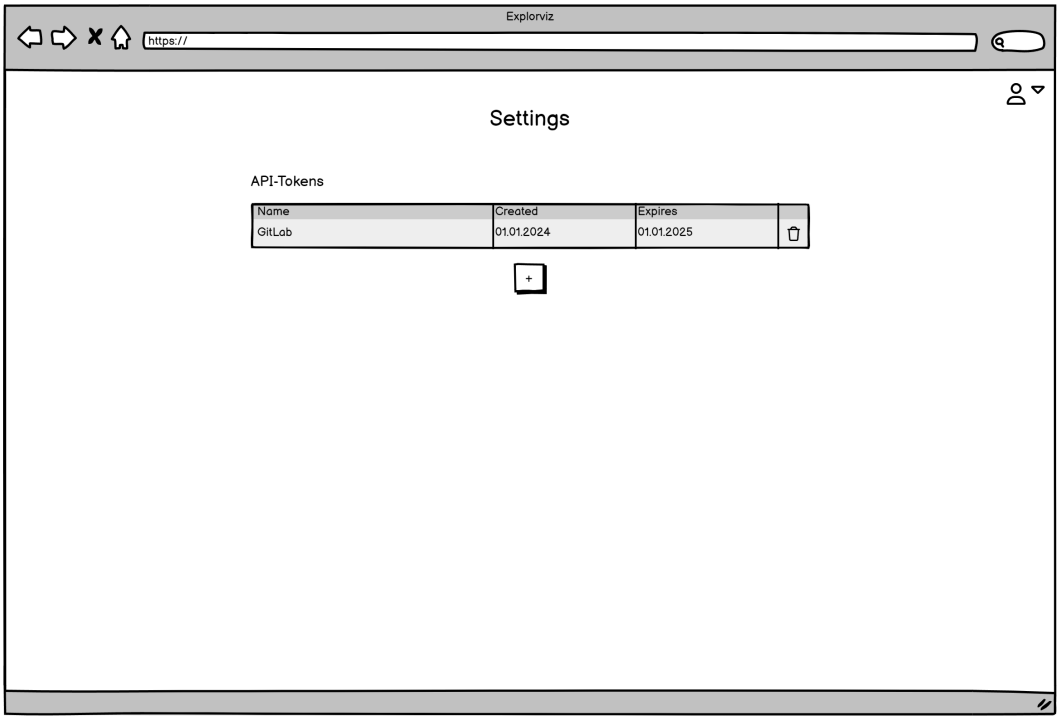


Abbildung 4.3. Mockup für eine Benutzerverwaltungsseite auf der API-Tokens von den Benutzern hinzugefügt werden können.

wird der Token selbst, sowie ein vom Benutzer gesetzter Name und die Git-Hosting Adresse gespeichert. Des Weiteren besteht die Option, ein Ablaufdatum für den API-Token zu setzten. Durch das Speichern von API-Tokens können diese vom Benutzer auf mehrere Issues angewendet werden und die Verwaltung wird erleichtert. Um zu der Benutzerverwaltung zu gelangen, wird das vorhandene Benutzer-Dropdown-Menü für die Benutzerinformationen um den Menüpunkt *Settings* erweitert. Dies ist in Abbildung 4.4 zu sehen. Über diesen Menüpunkt soll der Benutzer auf die Benutzerverwaltung weitergeleitet werden.

4.2.2. Erweiterung des *Restructuring*-Menüs

Die im Frontend vorhandene *Restructuring*-Komponente wird erweitert, um das Anlegen von Issues zu erleichtern. Außerdem soll es möglich sein, ein Issue mit einem *Snapshot* zu erstellen. Bevor der Benutzer ein Issue anlegen kann, muss über das Dropdown-Menü ein angelegter API-Token ausgewählt werden. Durch die Auswahl des API-Tokens, werden dem Benutzer alle GitLab-Projekte in einem weiteren Dropdown-Menü angezeigt. So soll

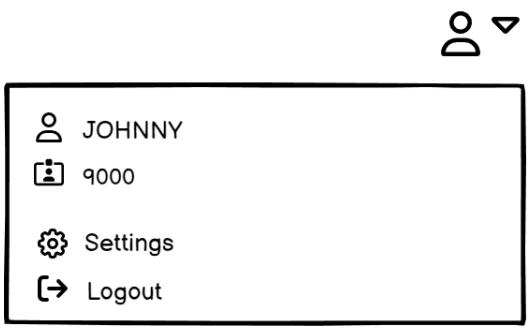


Abbildung 4.4. Mockup für das erweiterte Benutzer-Dropdown Menü.

es dem Benutzer erleichtert werden, ein Issue für ein Projekt anzulegen. Um in einem Issue einen *Snapshot* zu hinterlegen, wird eine *Snapshot*-URL erstellt, die automatisch in den Issue-Inhalt hinzugefügt wird. Dieser *Snapshot* wird außerdem auf der Startseite unter *Shared Snapshots* angezeigt. Es soll weiterhin die Möglichkeit bestehen, auch für den *Snapshot* ein Ablaufdatum anzugeben.

Erweiterung der Schnittstelle für Git-Hosting Services

ExplorViz nutzt die Git-Hosting Service Facade (*ghs-facade*) für den Zugang zu den Git-Hosting Services. Die Git-Hosting Service Facade ist ein weiterer Microservice, der auf die Git-Hosting Service APIs zugreift. Dieser Microservice soll erweitert werden, um über den API-Token dem Benutzer die Projekte anzuzeigen sowie die Erstellung eines Issues zu ermöglichen. Bisher wurde die Erstellung eines Issues nur im Frontend gehandhabt. Eine Auslagerung in eine Schnittstelle hat den Vorteil, dass auch andere Services die Git-Hosting Service Facade nutzen können.

4.3. Persistierung von Daten

Für die Persistierung der *Snapshot*-Daten sowie für die Persistierung der Benutzerdaten ist eine Datenbank notwendig. Der bereits vorhandene Microservice *User-Service* beinhaltet eine Schnittstelle zu einer MongoDB Datenbank. Dieser Microservice soll erweitert werden und die Verwaltung der *Snapshots* und Benutzerdaten in diesem Microservice stattfinden. Der genaue Ansatz sowie die Implementierung sind der Bachelorarbeit von Philipp Wulff [Wulff 2024] zu entnehmen. Die Persistierung ist auch ein wichtiger Bestandteil für die Funktionalität der in dieser Arbeit implementierten *Snapshots* und Git-Hosting Anbindung.

Kapitel 5

Implementierung

In diesem Kapitel wird die Implementierung der Persistierung der Ansicht einer Landschaft sowie die Git-Hosting Anbindung beschrieben. Der Fokus liegt dabei auf der Frontend-Entwicklung, die den wesentlichen Teil ausmacht. Um die Komplexität der Implementierung übersichtlich darzustellen, wird komponentenweise vorgegangen. Jede Komponente bzw. Komponentengruppe des Frontends erfüllt spezifische und visuelle Anforderungen und wird daher individuell betrachtet. Die Beschreibung des Frontends komponentenweise zu strukturieren, ermöglicht eine klare Darstellung der Implementierungsdetails. Jede Komponente bzw. Komponentengruppe wird in zwei Unterkapiteln betrachtet. Der Abschnitt visuelles Design widmet sich dabei dem Design der Komponente. Der Abschnitt technische Umsetzung geht auf die technische Realisierung der jeweiligen Komponenten ein.

5.1. Erweiterung der Startseite

Als Erstes werden wir über die Erweiterung der Startseite sprechen. Erklärt wird, welche Komponenten hinzugefügt wurden, welche Daten zur Verfügung gestellt werden müssen, und wie diese Daten geladen werden.

5.1.1. Visuelles Design

Die Startseite wurde, wie in Abbildung 5.1 zu sehen, um den Menüpunkt *Snapshots* erweitert. Innerhalb des Menüs kann aus den Unterpunkten *Personal Snapshots*, *Shared Snapshots* und *Subscribed Snapshots* gewählt werden. Als persönliche *Snapshots* sind die selbst erstellten *Snapshots* bezeichnet. Unter geteilte *Snapshots* sind einerseits die vom Benutzer geteilten persönlichen *Snapshots* zu finden, und andererseits die durch das Erstellen eines Issues generierten *Snapshots* eingeordnet. Wird dem Benutzer von einem anderen Benutzer ein geteilter *Snapshot* über einen Link zur Verfügung gestellt und wird dieser *Snapshot* darüber aufgerufen, so taucht dieser unter dem Unterpunkt *Subscribed Snapshots* auf. Dies soll dem Benutzer den Zugriff auf von anderen Benutzern geteilte *Snapshots* erleichtern.
Das Design der Tabellen wurde an die vorhandene Tabelle, die die Landschaft anzeigt, angepasst. Außerdem wurde die Funktion des Sortierens der Einträge durch Anklicken der Tabellenbeschriftung übernommen. Um eine *Snapshot*-Datei zu importieren, wurde

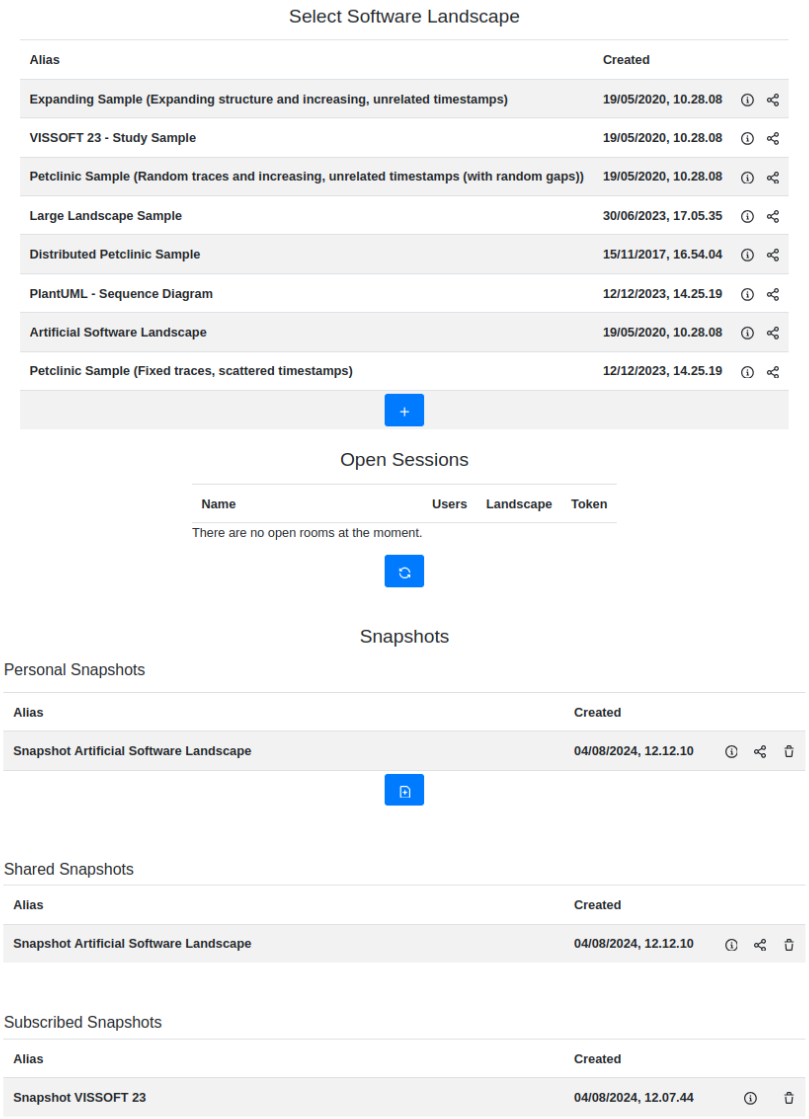
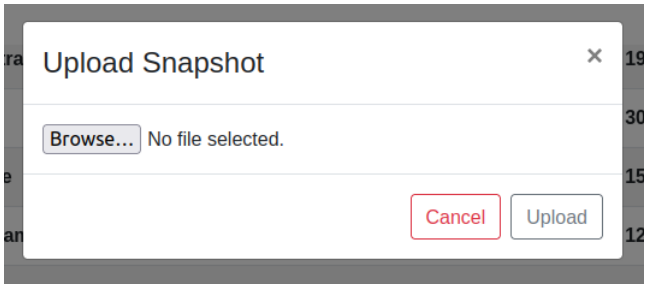
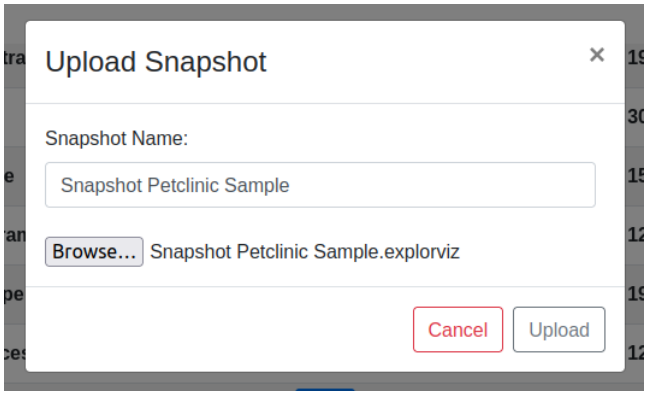


Abbildung 5.1. Die Startseite mit den hinzugefügten Tabellen für die verschiedenen *Snapshots*.

ein Button unterhalb der Tabelle platziert. Bei dem Design des Buttons wurde sich an dem vorherigen Buttondesign orientiert. Für das Design des Icons wurde sich für ein Datei-Import-Symbol entschieden, sodass das Icon dem Importieren einer *Snapshot*-Datei



(a) Das Fenster vor dem Auswählen einer Datei.



(b) Das Fenster, nach dem eine Datei ausgewählt wurde. Das Textfeld zeigt den Namen des *Snapshots* an und kann vom Benutzer geändert werden.

Abbildung 5.2. Das Fenster, welches sich öffnet, wenn ein *Snapshot* importiert wird. Das Fenster zeigt ein Textfeld mit dem Namen des *Snapshots* nach dem Auswählen einer *Snapshot*-Datei.

ähnelt. Es soll dem Benutzer symbolisieren, dass hier eine Datei importiert werden kann. Durch Anklicken des Buttons öffnet sich ein Fenster, in dem eine Datei ausgewählt werden kann. In Abbildung 5.2a ist dieses Fenster für einen Upload dargestellt. Wird eine Datei von dem Benutzer ausgewählt, wird ein Textfeld mit dem Namen des *Snapshots* angezeigt. Dieses Textfeld kann von dem Benutzer bearbeitet und beispielsweise der Name vor dem Upload geändert werden, bevor der *Snapshot* mit diesen Informationen gespeichert wird. Diese Ansicht ist in Abbildung 5.2b zu sehen.
Alle drei hinzugefügten Tabellen haben in der letzten Spalte verschiedene Icons. Das erste Icon zeigt die Metadaten an. Dieses Icon wurde aus der vorhandenen Tabelle übernommen. Es zeigt den Besitzer des *Snapshots* an, sowie den *Landscapetoken*, auf dem die Landschaft basiert. Ein *Landscapetoken* represäntiert eine Softwarelandschaft und besteht aus einem Schlüssel-Wert Paar, der von dem Backend generiert werden kann. Das letzte Icon, welches einen Mülleimer symbolisiert, ist auch in allen drei Tabellen verfügbar. Für persönliche *Snapshots* und geteilte *Snapshots* wird der *Snapshot* durch Anklicken des Icons gelöscht.

Für abonnierte *Snapshots* verschwindet durch Klicken des Mülleimer-Icons der *Snapshot* aus der Liste des Benutzers. Der *Snapshot* wird allerdings nicht gelöscht, da die Rechte zum Löschen des *Snapshots* nur sein Besitzer hat. Persönliche und geteilte *Snapshots* zeigen noch das Teilen-Icon. Wird dieses Icon bei einem persönlichen Icon angeklickt, wird von dem persönlichen *Snapshot* ein geteilter *Snapshot* erstellt sowie eine URL an das Clipboard gehängt, um den *Snapshot* aufzurufen. Wird das Icon bei einem schon geteilten *Snapshot* angeklickt, wird wiederum nur die URL an das Clipboard gehängt, da der geteilte *Snapshot* schon erstellt ist.

5.1.2. Technische Umsetzung

Bereitstellung der Daten

Um dem Benutzer alle wichtigen Informationen auf der Startseite anzeigen zu können, müssen alle Daten für die Startseite vorhanden sein. Um für eine Seite Daten zu laden, verwendet Ember.js den *model*-Hook in einer Route. In diesem Hook wird typischerweise eine API-Anfrage gemacht, um die benötigten Daten zu laden. Der *model*-Hook wird aufgerufen, wenn die Route betreten wird und die zurückgegebenen Daten werden dem Controller und dem Template zur Verfügung gestellt. Bisher wurden für die Startseite die *Landscapetokens* bereitgestellt. Um auch die *Snapshots* darzustellen, müssen auch diese Daten bereitgestellt werden. Daher wurde das Modell in ein Objekt mit zwei Attributen, *landscapeTokens* und *snapshotInfo*, erweitert. Beide Attribute erhalten durch eine API-Anfrage zum *User-Service* die relevanten Daten. Diese werden dann dem Controller und dem Template bereitgestellt und können durch *model.landscapeTokens* bzw. *model.snapshotInfo* aufgerufen werden.
Snapshots werden durch zwei verschiedene Objekte dargestellt. Das erste Objekt enthält alle Informationen über den *Snapshot* und wird für das Laden des *Snapshots* benötigt. Das andere Objekt enthält nur die Informationen, die benötigt werden, um alle Informationen auf der Startseite anzuzeigen. Je nach Anfrage stellt der *User-Service* eines der beiden Objekte bereit. Um den Datenverkehr zu verringern, wurde das *Snapshot*-Objekt in diese zwei Objekte unterteilt. Würden für jeden Aufruf der Startseite alle *Snapshots* mit ihren gesamten Daten bereitgestellt, könnte es zu Verzögerungen beim Laden der Startseite kommen. Da zudem nicht alle Informationen für das Anzeigen der *Snapshots* auf der Startseite relevant sind, reicht es, nur einen Teil der Daten zu versenden. Wenn ein *Snapshot* geladen werden soll, wird das komplette Objekt für nur diesen *Snapshot* übergeben. Dadurch wird der Datenverkehr verringert und auch bei einer großen Anzahl an *Snapshots* lädt die Startseite schnell.

Löschen und Teilen eines *Snapshots*

Um einen persönlichen, geteilten oder einen abonnierten *Snapshot* zu löschen, muss der Benutzer auf das Mülleimer-Icon des jeweiligen *Snapshots* klicken. Technisch wird dann

eine *DELETE*-Anfrage an den *User-Service* gestellt. Die Anfrage enthält die Informationen des *Snapshots*, sowie ob dieser ein geteilter *Snapshot* ist und ob es sich um einen abonnierten *Snapshot* handelt. Im *User-Service* wird dementsprechend der *Snapshot* gelöscht oder der Benutzer aus der Abonnentenliste des *Snapshots* entfernt.
Erstellt der Benutzer einen geteilten *Snapshot*, wird eine *PUT*-Anfrage an den *User-Service* gestellt. Es wird das Erstellungsdatum, optional das Ablaufdatum und der Besitzer des *Snapshots* an den *User-Service* versendet. Des Weiteren wird die URL des *Snapshot* erstellt, mit dem dieser geteilt werden kann. Die erstellte URL enthält alle Informationen, um den *Snapshot* zu laden und besteht aus dem *Landscapetoken*, dem Besitzer des *Snapshots* sowie dem Erstellungsdatum als ganze Zahl.

Importieren eines *Snapshots*

Wird ein *Snapshot* importiert, wählt der Benutzer eine Datei mit der Endung *.explorviz* aus. Die Datei wird im Frontend geparsed und als JSON eingelesen. Das JSON-Objekt gleicht dem *Snapshot*-Objekt und kann direkt als *Snapshot* gespeichert werden. Dabei wird mit dem JSON-Objekt eine API-Anfrage zu dem *User-Service* gemacht, welcher den *Snapshot* anlegt. Die API-Anfrage ist identisch zu der Anfrage, die beim Erstellen eines *Snapshots* verwendet wird.

5.2. Laden eines *Snapshots*

Als Nächstes beschäftigen wir uns mit dem Ladeprozess eines *Snapshots*. Welche Daten werden wie und wann geladen, und wie folgt die Darstellung eines *Snapshots*.

5.2.1. Visuelles Design

Die Darstellung eines geladenen *Snapshots* funktioniert wie die Darstellung einer Landschaft. Es wird nicht symbolisiert, dass es sich bei der geladenen Landschaft um einen *Snapshot* handelt. Der *Snapshot* ist nicht dazu gedacht, weitere Veränderungen an der Landschaft zu tätigen und automatisch zu speichern. Ein *Snapshot* dient nur der Speicherung der aktuellen Ansicht einer Landschaft, damit diese Ansicht wieder geladen werden kann. Daher ist es auch am Design nicht zu erkennen, dass es sich um einen *Snapshot* handelt. Möchte der Benutzer den *Snapshot* verändern, so muss nach dem Verändern des *Snapshots* ein neuer *Snapshot* erstellt werden.

5.2.2. Technische Umsetzung

Öffnet der Benutzer einen *Snapshot*, wird der Benutzer auf die Seite der Landschaften weitergeleitet. Dabei werden die Query-Parameter *createdAt*, *landscapeToken*, *owner* und *sharedSnapshot* gesetzt und die Route der *Landscapes* aktiviert. In der Route wird der Controller

aufgerufen mit dem die Seite initial gerendert wird. Bei dem Aufruf der Funktion werden nun alle Daten geladen, die für die Visualisierung benötigt werden. Die Query-Parameter sind so gesetzt, dass diese für eine API-Anfrage an den *User-Service* genutzt werden können, um die Daten des *Snapshots* zu erhalten. Anhand der Daten des *Snapshots* kann nun die Landschaft geladen werden. Als Nächstes werden die Strukturdaten und die dynamischen Strukturdaten der Landschaft sowie die Daten für den Zeitstrahl geladen. Der *Snapshot* hat all diese Daten selbst persistiert und beim Laden des *Snapshots* dürfen diese nicht neu geladen, sondern nur neu gesetzt werden. Beim Laden wird der Zeitstempel geprüft, um festzustellen, ob ein *Snapshot* existiert. Falls dies der Fall ist, wird der Zeitstempel direkt gesetzt, ohne diese neu aus dem *Span-Service* anzufragen. Beim Laden der Strukturdaten und der dynamischen Strukturdaten wird ebenfalls geprüft, ob ein *Snapshot* bereits existiert. Existiert dieser *Snapshot*, werden die Strukturdaten und die dynamischen Strukturdaten anstelle einer API-Anfrage zum Erhalt der Strukturdaten und dynamischen Strukturdaten an den *Span-Service*, gesetzt. Mit den gesetzten Strukturdaten des *Snapshots* wird die Landschaft aktualisiert. Bis zu diesem Zeitpunkt ist der Zeitstrahl sowie die Landschaft geladen worden. Nun muss nur noch die Ansicht auf den gespeicherten Stand aktualisiert werden. Hierzu wird als Hilfe das Persistieren eines Raumes genommen, welches durch die Kollaborationssessions bereits möglich ist. Der *roomSerializer* erhält den gespeicherten Raum von dem *Snapshot* und kann damit die Landschaft mit all den Hervorherbungen, Pop-Ups und Annotationen laden. Damit ist auch die Ansicht aktualisiert und zuletzt wird die Kameraposition des Benutzers auf die gespeicherte Kameraposition gesetzt.

5.3. Erweiterung der Anpassungsleiste um den Menüpunkt *Snapshot*

Um einen *Snapshot* zu erstellen, benötigt es eine Komponente, die den Benutzer einen *Snapshot* erstellen lässt. Es wird nun auf diese Komponente eingegangen und wie der Prozess für das Persistieren eines *Snapshot* abläuft.

5.3.1. Visuelles Design

Die Anpassungsleiste wurde um den Menüpunkt *Snapshot* erweitert. Befindet sich der Benutzer auf dem Menüpunkt, kann der Benutzer einen *Snapshot* von der aktuellen Landschaft erstellen oder eine *Snapshot*-Datei erstellen. Die Erweiterung der Anpassungsleiste ist in Abbildung 5.3 zu sehen. Um einen *Snapshot* zu erstellen, wählt der Benutzer einen Namen, danach kann der Benutzer einen *Snapshot* erstellen oder die aktuelle Ansicht als Datei exportieren. Die Buttons *Create Snapshot* und *Export Snapshot* sind so lange ausgegraut, bis ein Name gewählt wurde. Hat der Benutzer einen Namen gewählt, so werden die Buttons aktiviert. Das Ausgrauen der Buttons soll dem Benutzer darauf hinweisen, dass es zu diesem Zeitpunkt nicht möglich ist, einen *Snapshot* zu erstellen oder zu exportieren und

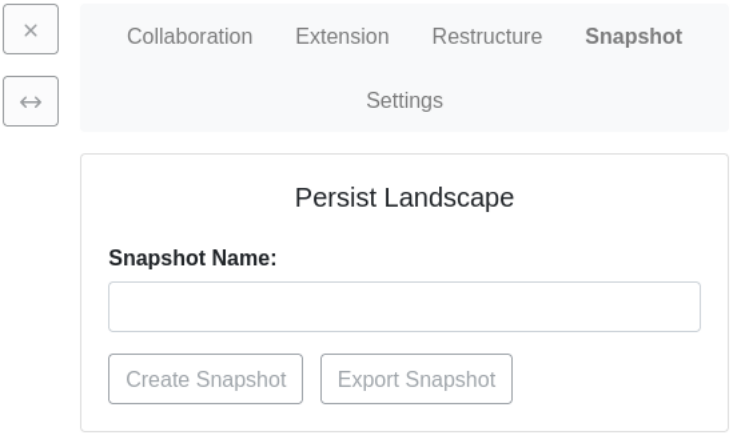


Abbildung 5.3. Die angepasste Anpassungsleiste zum Erstellen oder Exportieren eines *Snapshots*.

eine weitere Interaktion von dem Benutzer erwartet wird. Alternativ zur durchgeführten Erweiterung der Anpassungsleiste wäre es auch denkbar gewesen, per Rechtsklick einen *Snapshot* zu erstellen. Nachteilig daran wäre, dass ein weiteres Fenster geöffnet werden müsste, um den Namen des *Snapshots* festzulegen. Für den Benutzer ist es übersichtlicher, die Anpassungsleiste zu erweitern, da diese bereits aus mehreren Menüpunkten zur Ansicht besteht.

5.3.2. Technische Umsetzung

Damit der Benutzer einen *Snapshot* erstellen kann, müssen alle relevanten Daten in der Komponente für das Erstellen eines *Snapshots* vorhanden sein. Einige Daten sind durch die verschiedenen Services im Frontend verfügbar, andere Daten können bei der Erstellung erstellt werden oder sind durch Eingaben des Benutzers direkt verfügbar. Die restlichen Daten müssen über das Modell, beziehungsweise über den Controller bereitgestellt werden. Dazu zählen die Strukturdaten für eine Landschaft und der *Landscapetoken*. Um Zugriff auf die Strukturdaten und den *Landscapetoken* zu erhalten, müssen diese beim Laden der Visualisierung über den Controller bereitgestellt werden. Im Controller gesetzte Daten können im Template verwendet werden. Bei dem initialen Rendern der Landschaft werden die Strukturdaten, nachdem diese geladen worden sind, in dem Controller festgehalten und können im Template verwendet werden. Im Controller wird außerdem der *Landscapetoken* gesetzt. Das Template *visualization* reicht die Strukturdaten und den *Landscapetoken* als Attribut bis zu der Komponente für das Erstellen eines *Snapshots* weiter. Damit sind die Strukturdaten und der *Landscapetoken* beim Speichern eines *Snapshots* zur Verfügung gestellt. Die Annotationen sind durch den *AnnotationHandler*-Service verfügbar. Der Komponente für das Erstellen eines *Snapshots* werden die Annotationen als Attribut in der

browser-rendering-Komponente mit übergeben. Die Ansicht wird durch den *RoomSerializer*-Service gespeichert. Der *RoomSerializer*-Service hat die Methode *serializeRoom*, welche ein *SerializedRoom*-Objekt zurück gibt. In diesem Objekt sind alle relevanten Daten für eine Ansicht gespeichert. Dazu gehören die Pop-Ups, die Annotationen sowie die Hervorhebungen der Ansicht. Die Zeitstempel werden durch den Service *TimestampRepository* mit der Methode *getTimestamps* bereitgestellt. Die Kameraposition ist über den Service *LocalUser* verfügbar. Es werden hier die x-,y- und z-Koordinaten der Kameraposition gespeichert. Als letztes fehlen nur noch die Metadaten, welche direkt im Frontend erstellt werden. Der Benutzername wird als Eigentümer gesetzt, der Name des *Snapshots* ist durch die Eingabe des Benutzers gesetzt. Mit den gesamten Informationen wird das JSON-Objekt *SnapshotToken* erstellt, mit dem entweder eine Datei erstellt und diese zum Download dem Benutzer zur Verfügung gestellt wird oder eine API-Anfrage zu dem *User-Service* gestellt wird, um den *Snapshot* zu erstellen und zu persistieren.

5.4. Benutzerverwaltung

Bisher wurden nur die Komponenten für das Erstellen, Persistieren und Laden eines *Snapshots* beschrieben. In den nächsten Abschnitten wird es nun um die Erweiterung der Git-Hosting Anbindung gehen. Als Erstes beschäftigen wir uns mit der Erweiterung einer neuen Seite, um Benutzerdaten verwalten zu können. Außerdem wird darauf eingegangen, wie man zu der genannten Seite gelangt.

5.4.1. Visuelles Design

Die neu entstandene Seite ist vorgesehen für die Verwaltung genereller Benutzereinstellungen und für die Persistierung von Tokens. Mehrere API-Tokens können hinterlegt werden. Das Design orientiert sich an der Startseite. Die Tabelle der Benutzerverwaltung ist so aufgebaut, wie der Benutzer es von der Startseite kennt. Das Sortieren der Einträge durch Klicken der Tabellenbeschriftung wurde auch implementiert. Jeder Token ist mit einem Namen, dem Token selbst sowie dem Erstellungs- und Ablaufdatum gespeichert, wie es auf der Abbildung 5.4 zu sehen ist. In der letzten Spalte ist ein Icon mit einem Mülleimersymbol zu sehen. Klickt der Benutzer auf dieses Symbol wird der jeweilige Token gelöscht. Das Bearbeiten eines API-Tokens ist nicht möglich. Um einen API-Token zu erstellen, gibt es unter der Tabelle einen Button mit einem Plussymbol. Das Design des Buttons ist an das Design der Startseite angepasst. Wird der Button von dem Benutzer angeklickt, öffnet sich das in Abbildung 5.5 abgebildete Fenster. Für jeden API-Token muss der Benutzer einen Namen festlegen sowie den Token selbst angeben. Außerdem ist es notwendig, die URL der GitLab-Instanz anzugeben. Für die Host-URL ist ein Platzhalter als Beispiel eingeblendet. Optional kann der Benutzer ein Datum als Ablaufdatum des API-Token angeben. Das Ablaufdatum muss mindestens einen Tag in der Zukunft liegen.

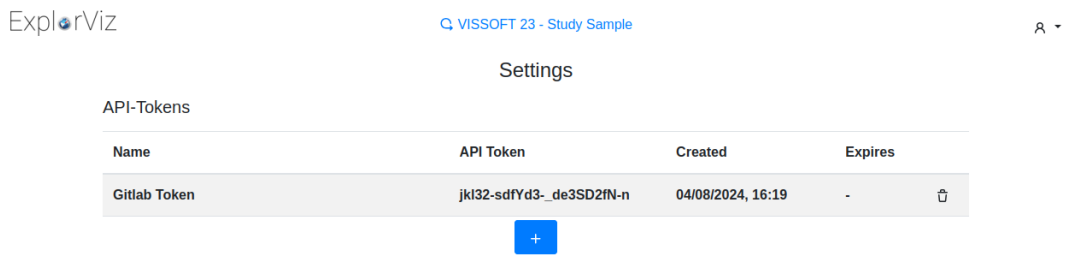


Abbildung 5.4. Die hinzugefügte Seite zur Verwaltung der Benutzereinstellungen und Tokens.

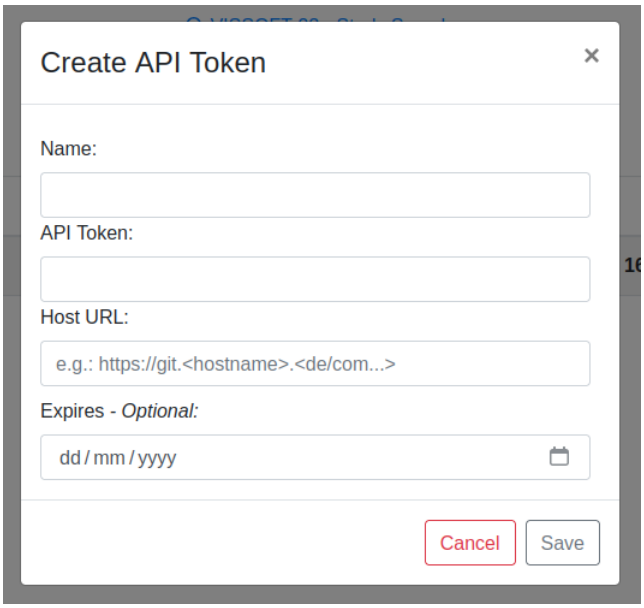


Abbildung 5.5. Für das Erstellen eines API-Tokens öffnet sich ein Fenster, damit die Angaben für den API-Token gemacht werden können.

Um die Benutzerverwaltungsseite aufzurufen, muss der Benutzer das Dropdownmenü über das Benutzersymbol öffnen. Das Dropdownmenü wurde um den Menüpunkt *Settings* erweitert, wie es in Abbildung 5.6 zu sehen ist. Durch Anklicken des Menüpunkts *Settings* wird der Benutzer auf die Benutzerverwaltungsseite weitergeleitet. Die Weiterleitung zu der Benutzerverwaltung über das Dropdownmenü wurde gewählt, da das Dropdownmenü an den Benutzer gebunden ist. Ein Benutzer erwartet, dass Benutzereinstellungen über den Benutzer gespeichert werden. Da es schon ein vorhandenes Dropdownmenü mit einem Benutzer-Icon gab, wurde das Dropdownmenü sinnhaft erweitert.

Abbildung 5.6. Das erweiterte Benutzer-Dropdownmenü mit dem neuen Unterpunkt *Settings*, der den Benutzer auf die neue Benutzerverwaltungsseite navigiert.

5.4.2. Technische Umsetzung

Routenführung

Für die Benutzerverwaltung wurde eine neue Seite erstellt. Um diese Seite zu erreichen, wird bei der Eingabe einer URL diese vom Router auf eine Route abgebildet. Diese Routen sind in einer Funktion abgebildet. Diese Funktion wurde erweitert, sodass die Route *settings* erreichbar ist. Bei der Eingabe einer URL mit dem Pfad *settings* wird nun der entsprechende Router-Handler aufgerufen und das Template und das Modell geladen. In dem Modell wird ein API-Aufruf an den *User-Service* durchgeführt. Das Modell erhält alle vom Benutzer gespeicherten API-Tokens, die auf der Seite angezeigt werden.

Erstellen eines API-Tokens

Wird ein neuer API-Token von dem Benutzer erstellt, müssen die Pflichtfelder von dem Benutzer ausgefüllt werden. Sind alle Pflichtfelder ausgefüllt, ist es möglich, einen API-Token zu erstellen. Für das Erstellen wird eine *POST*-Anfrage an den *User-Service* gestellt. Übergeben werden die Query-Parameter *uId*, welches die User-ID des Benutzers ist, sowie *name*, *token* und *hostUrl*, welche durch die Eingaben des Benutzers bekannt sind. Das Datum der Erstellung wird über den Query-Parameter *createdAt* übergeben. Optional wird ein Ablaufdatum als *expires* mitgegeben. Nach der API-Anfrage wird die Seite neu geladen, um den erstellten API-Token anzuzeigen.

Löschen eines API-Tokens

Soll ein API-Token gelöscht werden, wird eine API-Anfrage an den *User-Service* gestellt. Übergeben wird die User-ID des Benutzers (*uId*) und der API-Token (*token*). Nach der Anfrage wird die Seite analog zur Erstellung eines API-Tokens neu geladen. Der gelöschte API-Token wird durch das neu Laden des Modells der Route nicht mehr angezeigt.

5.5. Erweiterung des *Restructuring*-Menüs

Als Nächstes wird die Erweiterung des *Restructuring*-Menüs vorgestellt. Es soll darauf eingegangen werden, wie die Komponente erweitert und benutzerfreundlicher gestaltet wurde. Außerdem wird die neue Funktionsweise der Komponente erklärt.

5.5.1. Visuelles Design

In dem *Restructuring*-Menü war es bisher möglich, den API-Token, den GitLab-Issue Link und den GitLab Upload-Link zu hinterlegen. Diese Links wurden zum Erstellen eines Issues genutzt. Bisher war dem Benutzer nicht unbedingt klar, welche Links genau gebraucht werden, um einen Issue zu erstellen. Damit kann das Erstellen eines Issues direkt aus der Software heraus zu einer Hürde werden. Damit dem Benutzer das Erstellen eines Issues leichter fällt, wird die erste Maske durch Dropdownmenüs ersetzt, wie in Abbildung 5.7 zu sehen. Der API-Token wird durch das erste Dropdownmenü ausgewählt. Hier werden

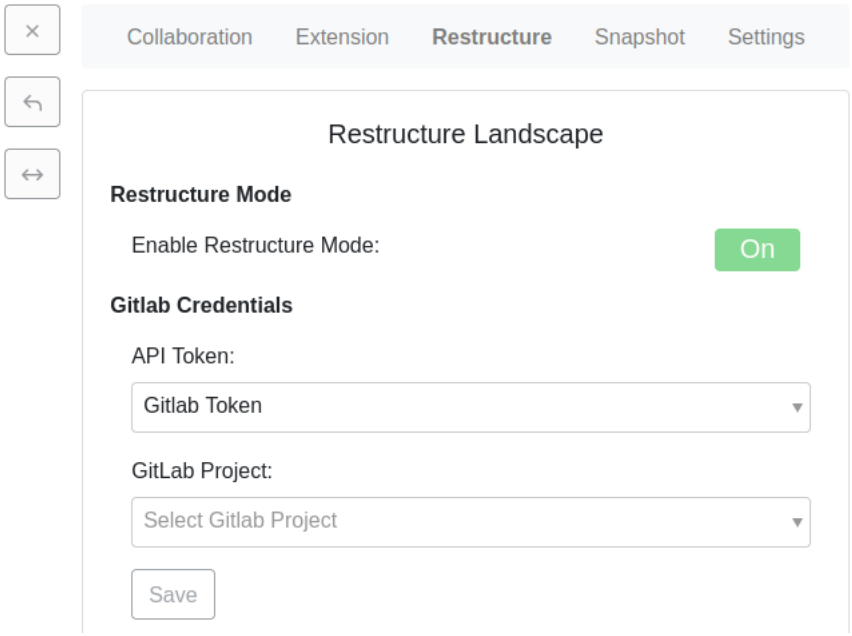


Abbildung 5.7. Die Erweiterung des *Restructuring*-Menüs zur Vereinfachung des Erstellens eines GitLab-Issue.

alle API-Token aufgelistet, die in der Benutzerverwaltung des Benutzers hinterlegt sind. Nachdem der API-Token ausgewählt wurde, kann der Benutzer über das zweite Dropdownmenü das Projekt auswählen, in dem das GitLab-Issue erstellt werden soll. Es werden

dem Benutzer alle Projekte angezeigt, die mit dem ausgewählten API-Token verfügbar sind. Solange der Save-Button ausgegraut ist, wird noch eine weitere Interaktion von dem Benutzer erwartet. Durch die beiden Dropdownmenüs soll es dem Benutzer vereinfacht werden, ein Projekt herauszusuchen. Das Ausprobieren verschiedener Links bei der Suche des richtigen Links wird damit umgangen.

Beim Erstellen eines Issues ist es dem Benutzer nun auch möglich, neben einem Screenshot auch einen *Snapshot* zu erstellen. Wie in Abbildung 5.8 zu erkennen, wurde ein weiterer Button hinzugefügt, mit dem der Benutzer ein *Snapshot* erstellen kann. Bei dem erstellten

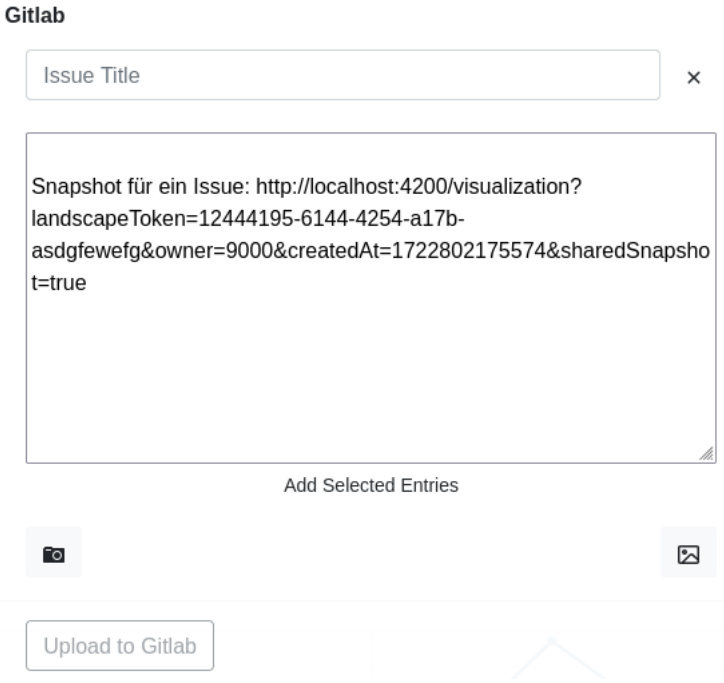


Abbildung 5.8. Erstellung eines Issues mit einem weiteren Button zur Erstellung eines *Snapshots*.

Snapshot handelt es sich um einen geteilten *Snapshot* und die *Snapshot*-URL wird automatisch in den Issuetext mit dem Snapshotnamen hinzugefügt. Der *Upload to GitLab* Button ist ausgegraut, bis der API-Token und das Projekt von dem Benutzer gesetzt worden sind. Der ausgegraute Button soll dem Benutzer zeigen, dass noch eine weitere Interaktion nötig ist. Wird ein *Snapshot* in der *Restructuring*-Maske erstellt, öffnet sich das in Abbildung 5.9 gezeigte Fenster. Der Benutzer muss dem *Snapshot* einen Namen geben und kann optional, da es sich um einen geteilten *Snapshot* handelt, ein Ablaufdatum setzen. Außerdem kann der Benutzer entscheiden, ob der *Snapshot* auch als Duplikat als persönlicher *Snapshot* gespeichert werden soll. Dazu kann der Benutzer das Kontrollkästchen markieren. Der

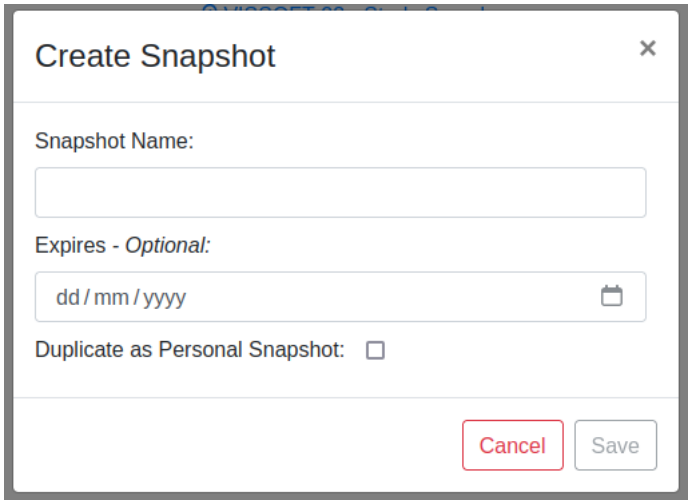


Abbildung 5.9. Erstellung eines Issues mit einem weiterem Button zur Erstellung eines *Snapshots*.

Save-Button ist wieder ausgegraut, bis der Benutzer alle verpflichtenden Angaben getätigt hat.

5.5.2. Technische Umsetzung

Auswählen der GitLab Zugangsdaten

Um einen GitLab-Issue zu erstellen, müssen zuerst die Zugangsdaten für GitLab und das Projekt ausgewählt werden. Mit dem obersten Dropdownmenü kann der Benutzer alle selbst angelegten API-Tokens einsehen und den gewünschten Token auswählen. Damit alle API-Tokens in dem Dropdownmenü angezeigt werden können, wird in dem initialen Rendern eine API-Anfrage an den *User-Service* gestellt, die alle API-Tokens des Benutzers zur Verfügung stellt. Das Rendern findet in dem Controller der Visualisierungsseite der Landschaften statt. Damit werden alle API-Tokens des Benutzers in dem Controller verfügbar und können im Template verwendet werden. Im Template werden die API-Tokens als Attribute bis zur *Restructuring*-Komponente weitergereicht. Damit sind alle API-Tokens in dem Dropdownmenü verfügbar und ein API-Token kann ausgewählt werden.
Nachdem ein API-Token ausgewählt wurde, kann der Benutzer ein Projekt auswählen. Um alle verfügbaren Projekte in dem Dropdownmenü anzuzeigen, muss mit dem ausgewählten API-Token eine API-Anfrage an die *ghs-facade* gestellt werden. Diese gibt alle mit dem API-Token verfügbaren Projekte zurück. Sobald der Benutzer das Dropdownmenü öffnet, wird diese asynchrone Anfrage gestellt, um alle Projekte anzuzeigen. Solange die Anfrage läuft, wird dem Benutzer die Nachricht, *Loading Projects...*, übermittelt.

Erstellung eines GitLab Issues

Für die Erstellung eines GitLab Issues wurde die meisten implementierten Funktionen übernommen. Es wurde die Funktion des Erstellens eines *Snapshots* hinzugefügt. Erstellt der Benutzer einen *Snapshot* aus dem *Restructuring*-Menü heraus, ist die Vorgehensweise identisch mit der im Abschnitt 5.3.2. Nach dem Erstellen wird der Issue-Inhalt um die *Snapshot*-URL und den Namen des *Snapshots* erweitert. Das Hochladen eines Issues wurde modifiziert, sodass es nun über den Service der *ghs-facade* gesteuert wird. Dafür wird eine API-Anfrage an die *ghs-facade* als *POST*-Anfrage gestellt, wobei ein JSON-Objekt mit der *Projekt-Id*, dem *API-Token*, der *Host-URL*, dem Titel und dem Inhalt des Issues mitgegeben wird.

5.6. Erweiterung der Schnittstelle für Git-Hosting Services

Zuletzt wird noch auf die Implementierung der Erweiterung der Schnittstelle für die Git-Hosting Services eingegangen. Dafür wurde der Microservice *ghs-facade* um die Erstellung eines Issues sowie die Bereitstellung der verfügbaren Projekten erweitert.

5.6.1. Technische Umsetzung

Die *ghs-facade* wurde um die folgenden API-Endpunkte erweitert:

- *⁄get_project*
- *⁄get_all_projects*
- */create_issue*

Die ersten beiden Endpunkte sind *GET*-Endpunkte, die jedoch als *POST*-Endpunkte verwendet werden. Grund dafür ist, dass ein JSON-Objekt mitgegeben werden soll, in dem die URL der GitLab-Instanz vorhanden ist. Dies wäre prinzipiell über einen Query-Parameter möglich, allerdings wird die API-Anfrage mit einem Query-Parameter, der eine weitere URL enthält, nicht richtig verarbeitet. Daher wurde der Endpunkt zu einem *POST*-Endpunkt abgeändert, um das Mitschicken von JSON-Objekten zu erlauben. Der Endpunkt */get_project* gibt ein Projekt zurück, wenn zusätzlich der Name des Projektes mitgeben wird. Der Endpunkt */get_all_projects* gibt alle mit dem mitgelieferten API-Token verfügbaren Projekte wieder. Beide Endpunkte geben die Projekte nur mit der ID und dem Namen wieder. Alle anderen Informationen werden vorher herausgefiltert. Der letzte Endpunkt ist für die Erstellung eines GitLab Issues zuständig.

Kapitel 6

Evaluation

In diesem Kapitel wird die Evaluation der Implementierung für die Umsetzung des Persistierens und Teilen einer Ansicht von einer Landschaft sowie die Erweiterung der Git-Hosting Anbindung präsentiert.

6.1. Ziele

Das Hauptziel dieser Evaluation ist es, die Benutzerfreundlichkeit und Funktionalität zu optimieren. Notwendig dafür ist die Erkenntnis, wie die Implementierung von verschiedenen Benutzern wahrgenommen wird. Daraus lassen sich notwendige Verbesserungen ableiten. Um die Benutzerfreundlichkeit und die Funktionalität der Implementierung zu evaluieren und unsere Ziele zu spezifizieren, werden folgende Forschungsfragen untersucht:

- Wie intuitiv ist die Benutzeroberfläche gestaltet?
- Wie sinnvoll empfinden die Benutzer die bereitgestellten Funktionen und Erweiterungen?
- Wie stabil und zuverlässig läuft die Implementierung im Gebrauch?

6.2. Methodik

Zur Beantwortung der Forschungsfragen haben wir uns für eine schriftliche Befragung entschieden. Die Probanden bekommen einen Fragebogen gestellt, in dem für jedes Feature eine offene Aufgabenstellung gestellt ist. Dazu kommen mehrere Fragen, die zu der gestellten Aufgabe beantwortet werden sollen. Die Aufgabenstellung beschreibt eine Aufgabe, die von den Probanden gelöst werden soll. Dabei wird pro Aufgabe ein Feature von den Probanden getestet. Zu jeder dieser Aufgaben sollen im Anschluss Fragen beantwortet werden. Bei den Fragen handelt es sich um geschlossene Fragen, welche vorgegebene Antwortmöglichkeiten enthalten. Zuletzt sind zu jedem Aufgabenteil zwei offene Fragen gestellt, bei denen die Probanden einen Freitext formulieren können. Während der Bearbeitung der Aufgaben stehen wir den Probanden für Verständnisfragen oder Hilfestellungen zur Verfügung. Wird Hilfestellung von den Probanden benötigt, so wird dies in dem Fragebogen für die jeweilige Aufgabe dokumentiert.

6.3. Experiment

Das Experiment ist in verschiedene Abschnitte unterteilt. Als Erstes wird auf den Versuchsaufbau eingegangen, um die Grundlagen des Versuches festzuhalten. Danach wird auf die Durchführung des Versuches eingegangen und zum Schluss wird der erstellte Fragebogen erläutert.

6.3.1. Versuchsaufbau

In dem folgenden Abschnitt werden wir den Versuchsaufbau beschreiben. Wir werden dabei auf die Probanden und deren Vorerfahrungen, auf die verwendete Hardware sowie auf die verwendete ExplorViz-Konfiguration und GitLab-Konfiguration, mit der die Probanden arbeiten werden, eingehen.

Probanden

Für die Evaluation wurden Probanden mit unterschiedlichem Hintergrundwissen ausgewählt, um ein breites Spektrum an Nutzererfahrungen abzudecken. Die Mehrheit der Probanden stammte aus dem Informatikbereich, wodurch uns eine fundierte Einschätzung der technischen Aspekte der Implementierung ermöglicht wurde. Gleichzeitig wurden auch Probanden ohne spezifisches Hintergrundwissen im Bereich Informatik einbezogen, um die Benutzerfreundlichkeit für weniger technikaffine Anwender zu evaluieren. Diese diverse Auswahl an Probanden gewährleistet eine umfassende und ausgewogene Bewertung der Implementierung hinsichtlich ihrer Benutzerfreundlichkeit und Funktionalität.

Hardware

Für die Evaluation standen zwei Laptops mit jeweils einem externen Monitor und einer externen Maus zur Verfügung. Die Tabelle 6.1 zeigt die Spezifikationen der beiden Laptops.

Tabelle 6.1. Die verwendete Hardware mit der die Evaluation durchgeführt wurde.

	Laptop 1	**Laptop 2**
Modell	Lenovo Yoga Pro 7 14IRH8	ASUSTeK COMPUTER INC. UX331UN
Prozessor	Intel Core i7-13700H 3.7GHz x 14	Intel Core i7-8550U 1.80GHz x 8
Arbeitsspeicher	32 GB	16 GB
Grafikkarte	Intel Iris Xe Graphics	Nvidia GeForce MX150
Speicherplatz	512 GB SSD	512 GB SSD
Betriebssystem	Ubuntu 22.04.4 LTS 64-bit	Ubuntu 22.04.4 LTS 64-bit

ExplorViz-Konfiguration

Um alle Funktionen für ExplorViz bereitzustellen, wurde das Frontend mit dem dazugehörigen Docker-Container lokal ausgeführt. Für die erweiterten Features wurden eine MongoDB in einem Docker-Container gestartet und der *User-Service* lokal auf dem Laptop ausgeführt. Zudem wurde der *Collaboration-Service* und die *ghs-facade* lokal auf dem Laptop gestartet. Damit arbeiteten alle Probanden mit derselben ExplorViz-Version und die Evaluation lief unter den gleichen Bedingungen ab. Um eine identische Ausgangslage für alle Probanden zu gewährleisten, wurde für die Evaluation nur in der Softwarelandschaft *VISSOFT 23 - Study Sample* des Demo-Suppliers gearbeitet, welche in Abbildung 6.1 zu sehen ist.

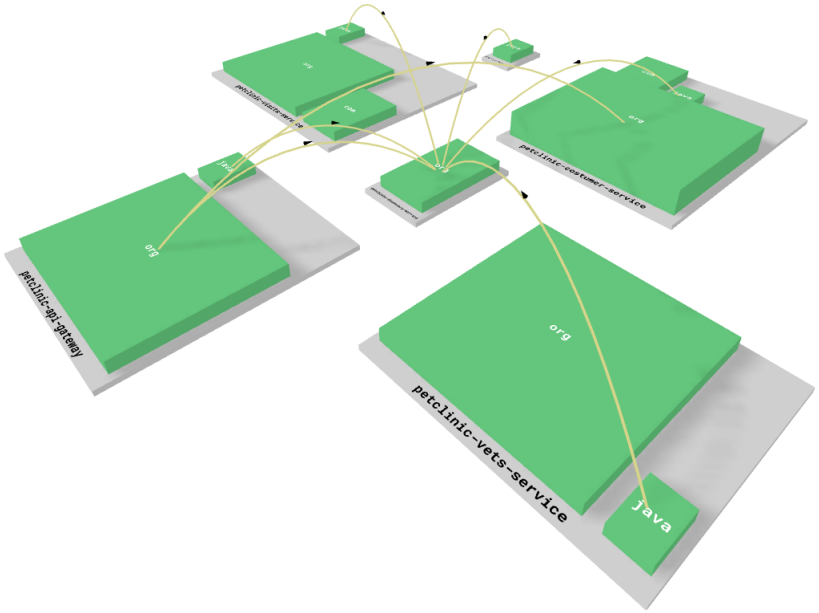


Abbildung 6.1. Ein Screenshot von ExplorViz, welches die *VISSOFT 23 - Study Sample* als Beispielanwendung visualisiert.

GitLab Konfiguration

Jeder Proband hat für die Erstellung eines Issues einen GitLab-API-Token, die zugehörige URL und ein Projekt erhalten, für das ein Issue erstellt werden soll. Alle Probanden haben dasselbe Projekt und denselben GitLab-API-Token erhalten.
Vor jedem Versuch wurde für jeden Probanden ein geteilter *Snapshot* als Issue für das Projekt erstellt. Dieser *Snapshot* wurde von einer anderen ExplorViz-Benutzer-Id erstellt als

die Benutzer-Id der Probanden. Der *Snapshot* musste nicht bei jedem Probanden identisch sein, da der Inhalt des *Snapshots* für die Aufgabenstellung ohne Relevanz war.

6.3.2. Versuchsdurchführung

Die Durchführung der Versuche erfolgte in mehreren strukturierten Schritten, um eine konsistente Evaluation zu gewährleisten. Zu Beginn wurden die Probanden in die Testumgebung eingeführt. Abhängig von dem Wissensstand der Probanden wurde eine detaillierte oder kurze Einführung in ExplorViz gegeben. Für Probanden ohne spezifisches Hintergrundwissen in ExplorViz oder im Bereich der Informatik gab es eine kurze Einleitung in das Thema der Softwarevisualisierung und dessen Sinn. Danach gab es eine detaillierte Einführung in ExplorViz, bei der alle für die Evaluation benötigten Funktionen sowie weitere, für das Verständnis von ExplorViz wichtigen Funktionen gezeigt wurden. Nach der Einweisung sollten die Probanden unter Beaufsichtigung ExplorViz selbst testen und aufkommende Fragen stellen. Für Probanden, die bereits Erfahrungen mit ExplorViz aufweisen, wurde die Einleitung kurz gehalten. Anschließend wurden die implementierten Funktionen grob beschrieben, damit die Probanden den Hintergrund der Evaluation verstehen.
Die Probanden erhielten nach der Einführung den Fragebogen mit den zu bearbeitenden Aufgaben und Fragen. Während des gesamten Versuchs konnte bei Bedarf Hilfestellung durch uns gegeben werden. Dies wurde im Falle dokumentiert.

6.3.3. Fragebogen

Im Folgenden wird der eingesetzte Fragebogen detailliert beschrieben. Zu Beginn werden personenbezogene Daten abgefragt. Der Fragebogen ist bei jeder getesteten Komponente in gleicher Form aufgebaut. Zuerst wird den Probanden eine zu bearbeitende Aufgabe gestellt. Nach Bearbeitung oder Abschluss der Aufgabe werden Fragen zu dieser gestellt. Die erste Frage dokumentiert eine eventuell gegebene Hilfestellung. Die zwei letzten, offenen Fragen geben dem Probanden die Möglichkeit, ein generelles Feedback zu der jeweiligen Anwendung zu geben. In der Evaluation gibt es insgesamt vier Aufgaben mit jeweils einem Frageteil. Für diese Arbeit sind nur drei der vier Aufgaben relevant.

Personenbezogene Daten

Zunächst werden personenbezogene Daten der Probanden abgefragt. Zur Einschätzung des Hintergrundwissens wurden die Probanden nach ihrer ausgeübten Tätigkeit gefragt. Dabei stand Folgendes zur Auswahl: *Student (Informatik), Student (Nicht-Informatik), Wissenschaftlicher Mitarbeiter (Informatik), Wissenschaftlicher Mitarbeiter (Nicht-Informatik), Mitarbeiter in der Informatik, Mitarbeiter außerhalb der Informatik, Sonstiges*. Außerdem wird der höchste Bildungsabschluss abgefragt. Als Letztes wird nach den Erfahrungen in den Bereichen zu ExplorViz, Software-Architektur, Softwarevisualisierungstools, Softwareentwicklung,

Webentwicklung, Design, objektorientierte Programmierung und GitLab gefragt. Die Probanden müssen ihren eigenen Erfahrungsstand beurteilen und wählen zwischen den Auswahlmöglichkeiten *Sehr viel, Viel, Moderat, Wenig, Sehr Wenig* und *Keine* aus.

Benutzerverwaltung

Die erste Aufgabe testet die neu implementierte Benutzerverwaltung. Die Probanden sollen auf die neue Seite navigieren und zwei neue API-Tokens anlegen. Der eine API-Token ist von uns vorgegeben und der andere ist frei zu wählen. Danach soll der frei gewählte API-Token gelöscht werden. Nach der Bearbeitung der Aufgabe sollen die Probanden die Benutzerfreundlichkeit auf einer Skala von 1-5 bewerten, wobei 1 als sehr intuitiv und 5 als sehr kontraintuitiv gilt. Die Fragen sind in der Tabelle 6.2 abgebildet.

Tabelle 6.2. Die gestellten Fragen zu der Benutzung der neuen Benutzerverwaltung.

ID	Fragen
C1	Hat das Anlegen eines Tokens ohne Hilfestellung funktioniert?
C2	Wie intuitiv fanden Sie das Erstellen eines API-Tokens?
C3	Wie intuitiv fanden Sie das Löschen eines API-Tokens?
C4	Wie intuitiv fanden Sie die Navigation zu den Benutzereinstellungen?

Snapshots

In dieser Aufgabe sollen die Probanden mit der neuen Erweiterung zu den *Snapshots* arbeiten. Die Probanden arbeiten in der Landschaft *VISSOFT 23 - Study Sample*. Als erstes soll die Ansicht bearbeitet, dann ein *Snapshot* erstellt und ein *Snapshot* exportiert werden. Im Anschluss soll der erstellte *Snapshot* geteilt und der exportierte *Snapshot* importiert werden. Danach sollen alle *Snapshots* gelöscht werden. Der von uns bereitgestellte *Snapshot* soll als Letztes aufgerufen werden. Nach der Bearbeitung der Aufgabe beantworten die Probanden die Fragen zu der Benutzerfreundlichkeit und Funktionalität, welche in Tabelle 6.3 dargestellt sind. Die Antwortmöglichkeiten sind in einer Skala von 1-5 dargestellt, wobei 1 die bestmögliche und 5 die schlechteste Antwort darstellen.

Erweiterung der Git-Hosting Anbindung

Die letzte Aufgabe befasst sich mit der Erweiterung der Git-Hosting Anbindung. Die Probanden sollen ein GitLab-Issue direkt aus ExplorViz erstellen. Dabei soll das Issue einen *Snapshot* und einen Screenshot enthalten. Der Titel und ein Issue-Text können frei gewählt werden. Die Probanden sollen auch hier nach dem Bearbeiten der Aufgabe die Benutzerfreundlichkeit auf einer Skala von 1-5 bewerten, wobei 1 sehr intuitiv und 5 sehr kontraintuitiv ist. Die Fragen sind in Tabelle 6.4 aufgelistet.

Tabelle 6.3. Die gestellten Fragen zu der Benutzung der neuen Snapshotfunktionen.

ID	Fragen
E1	Hat das Erstellen eines *Snapshots* ohne Hilfestellung funktioniert?
E2	Wie intuitiv fanden Sie das Erstellen eines *Snapshots*?
E3	Wie intuitiv fanden Sie das Exportieren eines *Snapshots*?
E4	Wie intuitiv fanden Sie das Importieren eines *Snapshots*?
E5	Wie intuitiv fanden Sie das Öffnen eines *Snapshots*?
E6	Wie intuitiv fanden Sie das Teilen eines *Snapshots*?
E7	Wie intuitiv fanden Sie das Löschen eines *Snapshots*?
E8	Wie gut fanden Sie die Navigation für die Snapshotfunktion?
E9	Wie sinnvoll empfinden Sie die Funktion des *Snapshots*?

Tabelle 6.4. Die gestellten Fragen zu der Benutzerfreundlichkeit bei dem Erstellen eines GitLab-Issues.

ID	Fragen
F1	Hat das Erstellen eines Issues mit URL ohne Hilfestellung funktioniert?
F2	Wie intuitiv war die Navigation zu der Issue Erstellung?
F3	Wie intuitiv empfanden Sie die Auswahl des Tokens und der Projekte?
F4	Wie intuitiv empfanden Sie den Aufbau der *Restructuring*-Maske?
F5	Wie intuitiv fanden Sie die Erstellung eines *Snapshots* im *Restructuring*-Menü?

6.4. Ergebnisse

In diesem Abschnitt werden die Ergebnisse der Evaluation präsentiert. Die Ergebnisse der geschlossenen Fragen werden gesammelt präsentiert. Die Ergebnisse der offenen Frage werden nicht gezeigt, da die vielen unterschiedlichen Antworten zu umfangreich für diese Arbeit wären. Alle Ergebnisse sind öffentlich verfügbar [Wulff und Brehme 2024].

Probanden

An der Evaluation haben 14 Probanden teilgenommen, davon 13 Studierende und ein Wissenschaftlicher Mitarbeiter in der Informatik. 3 Studierende kommen nicht aus dem Informatikbereich. Knapp 60% der Probanden haben bisher keine Erfahrung mit ExplorViz, die restlichen Probanden haben viel oder moderate Erfahrung mit ExplorViz. Es gibt drei Probanden, die keine Erfahrung mit Softwareentwicklung haben und 2 Probanden, die weder Erfahrungen in der Webentwicklung noch in der objektorientierten Programmierung haben. Die Mehrzahl wies aber ein generelles Verständnis von Programmierung auf. Die Kenntnis über Softwarearchitektur unter den Probanden ist ähnlich verteilt wie die Vorerfahrungen in der Programmierung. Es gibt drei Probanden ohne jegliche Kenntnis, die restlichen Probanden brachten eine generelle Kenntnis über Softwarearchitektur mit. Softwarevisualisierungstools wurden von den meisten Probanden bisher wenig genutzt

und Erfahrungen mit Design war ebenfalls wenig vorhanden. Bis auf 3 Probanden, die keine Erfahrung mit GitLab haben, wiesen alle Probanden einen sehr guten Kenntnisstand darüber auf.

6.4.1. Benutzerverwaltung

Benutzerfreundlichkeit

Die Ergebnisse für die Benutzerfreundlichkeit der neu angelegten Benutzerverwaltung sind in der Tabelle 6.5 zu sehen. Zusätzlich zu dem arithmetischen Mittel wurde die Standardabweichung als Indikator für die Verteilung der Antworten angegeben. 12 der 14 Probanden haben diese Aufgabe ohne Hilfestellung lösen können.

Tabelle 6.5. Die Resultate der Benutzerfreundlichkeit für die neu angelegte Benutzerverwaltung.

ID	Arithmetisches Mittel	Standardabweichung
C2	1.57	0.76
C3	1.07	0.27
C4	2	0.88

6.4.2. *Snapshots*

Benutzerfreundlichkeit

Die Resultate für die Benutzerfreundlichkeit der neu implementierten Snapshotfunktionen sind in der Tabelle 6.6 präsentiert. Diese Aufgabe wurde von 13 der 14 Probanden ohne Hilfestellung gelöst.

Tabelle 6.6. Die Resultate der Benutzerfreundlichkeit für die neu angelegte Benutzerverwaltung.

ID	Arithmetisches Mittel	Standardabweichung
E2	1.57	0.76
E3	1.79	1.05
E4	1.36	0.84
E5	1.21	0.58
E6	1.21	0.58
E7	1	0
E8	1.93	0.83

Funktionalität

Zuletzt wurden die Probanden über die Sinnhaftigkeit der neuen Snapshotfunktion gefragt. Die Probanden sollten die Sinnhaftigkeit der Snapshotfunktion auf einer Skala von 1-5 einschätzen. 1 galt dabei als sehr sinnvoll und 5 als nicht sinnvoll. Das Resultat ist in der Tabelle 6.7 präsentiert.

Tabelle 6.7. Das Resultat über den Nutzen der Snapshotfunktion.

ID	Arithmetisches Mittel	Standardabweichung
E9	1.21	0.58

6.4.3. Erweiterung der Git-Hosting Anbindung

Benutzerfreundlichkeit

Die Resultate für die Benutzerfreundlichkeit von der Erweiterung der Git-Hosting Anbindung sind in der Tabelle 6.8 zu sehen. 9 von 14 Probanden haben diese Aufgabe ohne Hilfestellung gelöst.

Tabelle 6.8. Das Resultat über die Erweiterung der Git-Hosting Anbindung.

ID	Arithmetisches Mittel	Standardabweichung
F2	3	1.3
F3	1.86	1.23
F4	2.5	1.16
F5	1.79	1.31

6.5. Diskussion der Ergebnisse

In diesem Abschnitt werden die Ergebnisse der Evaluation diskutiert. Anhand der drei verschiedenen Erweiterungen werden jeweils die relevanten Forschungsfragen beantwortet.

6.5.1. Benutzerverwaltung

Als Erstes betrachten wir die neu hinzugefügte Seite, die für die Benutzerverwaltung gedacht ist. Dabei werden wir auf die Benutzerfreundlichkeit der Funktionen und der Navigation sowie auf die Funktionalität eingehen.

Benutzerfreundlichkeit

Die Resultate der Evaluation für die Benutzerverwaltung war im Ganzen positiv. 12 der 14 Probanden konnten die Aufgaben zu der Benutzerverwaltung ohne Hilfestellung lösen. Dies spricht für eine Benutzerverwaltung, in der ein Benutzer sich leicht in die neue Erweiterung einarbeiten kann. Das Erstellen und Löschen von API-Tokens stellte für die Probanden auch kein Problem dar. Da hier eine zur Startseite ähnliche Ansicht gewählt wurde, kamen vor allem erfahrene ExplorViz-Benutzer sehr gut zurecht. Durch die allseits bekannte Assoziation eines Mülleimers mit dem Löschvorgang bereitete das Mülleimersymbol den Probanden beim Löschen kein Problem. Als einzige Schwachstelle stellt sich die Navigation zu der Benutzerverwaltung heraus. Hier war es für einige Probanden nicht ersichtlich, über das Benutzericon zu der Seite zu navigieren. Die beiden Probanden, die Hilfestellung benötigt haben, haben für die Navigation die Hilfestellung verwendet. Dabei scheint das Hintergrundwissen der Probanden von wenig Relevanz. Probleme traten sowohl bei Probanden mit Erfahrung in ExplorViz, die demnach mit dem Benutzerdropdown vertraut waren, als auch bei Probanden ohne derartige Vorerfahrung auf.

Funktionalität

Die Probanden hatten im Allgemeinen keine Schwierigkeiten beim Anlegen und Löschen von API-Tokens. Diese Funktionen funktionierten zuverlässig. Im Rahmen des Feedbacks wurde eine Unstimmigkeit in der Funktionalität des Ablaufdatums angegeben. Bei dem Anlegen eines API-Tokens kann ein Ablaufdatum entweder über den Date-Picker oder über eine direkte Eingabe von Zahlen über die Tastatur erstellt werden. Erfolgt die Eingabe über den *Date-Picker*, ist das Erstellen eines Datum in der Vergangenheit nicht möglich. Erfolgt die Eingabe über die Tastatur direkt, wird dagegen keine Überprüfung des Datums durchgeführt. Damit ist es möglich, ein API-Token zu erstellen, welcher ein Ablaufdatum in der Vergangenheit hat. Der API-Token kann erfolgreich erstellt werden, wird aber dem Benutzer nicht angezeigt, da der Token bereits abgelaufen ist.

6.5.2. *Snapshots*

In diesem Abschnitt geht es um die Ergebnisse der Evaluation über die *Snapshots*. Dabei wird in Benutzerfreundlichkeit und Funktionalität unterteilt. Allgemein waren die Ergebnisse lobenswert.

Benutzerfreundlichkeit

13 von 14 Probanden haben die Aufgaben ohne Hilfestellung bearbeiten können. Die Navigation, um einen *Snapshot* zu erstellen, wurde von den Probanden immer schnell gefunden. Der erste Ansatz der Probanden ging meist über das Menü, wo auch das Erstellen eines *Snapshots* möglich ist. Dementsprechend sind auch die Resultate für das Erstellen eines

Snapshots positiv. Über die Menümaske ist auch das Exportieren eines *Snapshots* möglich. Die Probanden haben auch hier schnell einen Export erstellen können. Verwirrungen gab es, da ein *Snapshot* nicht gespeichert und im selben Schritt exportiert werden kann. Ein *Snapshot* kann nach dem Erstellen entweder exportiert oder gespeichert werden.
Öffnen, Löschen und Teilen eines *Snapshots* stellte für die Probanden keine Schwierigkeit dar. Die Probanden fanden schnell die Funktionen und hatten auch mit der Verwendung keine Probleme. Damit ist es neuen Benutzern schnell möglich, sich in die Erweiterung einzuarbeiten und diese zu verwenden.
Die Benutzeroberfläche ist für das Erstellen und Exportieren eines *Snapshots* verständlich aufgebaut und ein Benutzer findet schnell die Funktionen, wenn eine Ansicht einer Landschaft geöffnet ist. Um einen *Snapshot* zu teilen, zu löschen oder zu öffnen, muss der Benutzer zurück auf die Startseite gelangen. Die Probanden suchten oft weiter in der Menüleiste nach den *Snapshots*, bevor sie als Zweitidee auf die Startseite zurück navigierten. Befand sich der jeweilige Benutzer auf der Startseite, war es kein Problem, die weiteren Funktionen zu nutzen. Daher sind die Ergebnisse für das Navigieren der Snapshotfunktion auch generell positiv. Im Gegensatz zu den anderen Funktionen lagen die Antworten überwiegend bei intuitiv und nicht bei sehr intuitiv.

Funktionalität

Das Erstellen und Exportieren eines *Snapshots* hat bei allen Probanden funktioniert und auch das Aufrufen des *Snapshots* war ohne Probleme möglich. Keiner der Probanden berichtete uns von Problemen oder dass *Snapshots* nicht richtig geladen wurden. Das Teilen des *Snapshots* funktioniert und die Probanden konnten den *Snapshot* über den Abruflink wieder aufrufen. Die Funktionen des *Snapshots* funktionierten alle zuverlässig. Außerdem wurde die Funktion des *Snapshots* als sehr sinnvoll von den Probanden eingestuft.

6.5.3. Erweiterung der Git-Hosting Anbindung

In diesem Abschnitt wird die Erweiterung der Git-Hosting Anbindung betrachtet. Auch hier wird auf die Benutzerfreundlichkeit und die Funktionalität eingegangen.

Benutzerfreundlichkeit

Die Probanden empfanden die *Restructuring*-Komponente nicht intuitiv. 5 von 14 Probanden benötigten Hilfestellung, um die Aufgabe des Erstellens eines Issues zu absolvieren. Dies spricht für eine nicht ideale Benutzerfreundlichkeit. Dabei wurde von den Probanden der allgemeine Aufbau des *Restructuring*-Menüs bemängelt und als nicht intuitiv empfunden. Oft wussten die Probanden nicht, warum sich die Funktion zur Erstellung eines Issues in dem *Restructuring*-Menü befindet. Es wurde als Feedback vorgeschlagen, die Erstellung eines Issue als eigenständiges Menü zu implementieren. Die Token- und Projektauswahl über das Dropdownmenü wurde dagegen als intuitiv bewertet und die Probanden hatten

auch keine Probleme, die Auswahl zu tätigen. Die Erstellung eines Issues, in dem ein *Snapshots* hinzugefügt wird, wurde von den Probanden als intuitiv bewertet.

Funktionalität

Die Erstellung eines Issues mit einem *Snapshot* hat bei allen Probanden gut und zuverlässig funktioniert. Es wurde uns von keinem der Probanden mitgeteilt, dass die Erstellung fehlgeschlagen sei. Ein Proband hat einen Fehler beschrieben, welcher mit dem Speichern des Tokens und Projektes zu tun hat. Hier war im Cache noch ein API-Token gespeichert, der im Dropdownmenü angezeigt wurde, obwohl der Token gar nicht mehr existierte.

6.6. Validität der Ergebnisse

Die Validität der Ergebnisse der Evaluation wird durch mehrere Faktoren beeinflusst, die im Folgenden erläutert werden.

Probanden

Insgesamt nahmen 14 Probanden an der Evaluation teil und die Zusammensetzung der Testgruppe war recht heterogen. Diese heterogene Zusammensetzung der Testgruppe ermöglicht zwar einen Einblick in die Benutzerfreundlichkeit und Funktionalität der Implementierung, doch durch die relativ geringe Gesamtanzahl an Probanden kann eine Generalisierung der Ergebnisse nur eingeschränkt erfolgen. Hinzukommend waren die meisten Probanden aus dem Freundeskreis der Versuchsleitung, wodurch eine Verzerrung der Bewertung nicht ausgeschlossen werden kann.
Des Weiteren ist zu beachten, dass acht der 14 Probanden keine Erfahrung in ExplorViz vorwiesen und damit lediglich Benutzer repräsentieren, die sich neu in ExplorViz einarbeiten. Diese Probanden geben wichtige Einblicke in die Zugänglichkeit von ExplorViz, repräsentieren aber nicht die gesamte Zielgruppe der erfahrenen Anwender.

Einführung in ExplorViz

Die Einführung in ExplorViz wurde so konsistent wie möglich durchgeführt, doch gab es schon aufgrund verschiedener Wissensstände der Probanden Variationen. Diese Variabilität könnte die Ergebnisse beeinflusst haben, da die Probanden möglicherweise auf unterschiedliche Aufgaben unterschiedlich gut vorbereitet waren.

Einfluss der Unterstützung

Während die Probanden die Aufgaben bearbeitet haben, standen wir für Fragen zur Verfügung. Diese Unterstützung kann die Ergebnisse beeinflusst haben, da die Probanden möglicherweise durch diese Hilfestellung auf weniger Schwierigkeiten gestoßen sind, als

wenn sie auf sich allein gestellt gewesen wären. Dadurch kann die wahrgenommene Benutzerfreundlichkeit besser ausgefallen sein als unter realen Bedingungen ohne Hilfestellung.

Kapitel 7

Verwandte Arbeiten

In diesem Kapitel wird es einen Überblick über bestehende Forschungsarbeiten geben, die in Zusammenhang mit dem vorliegenden Thema stehen.

7.1. Source Viewer 3D

Die Arbeit von [Marcus u. a. 2003] beschäftigt sich mit der Entwicklung von Source Viewer 3D (sv3D), einem Tool, das die Visualisierung von Softwaresystemen ermöglicht. Ein zentraler Aspekt des Designs von sv3D ist die Trennung der Visualisierung von der Datenerhebung. Um diese Flexibilität zu gewährleisten, wurde das Eingabeformat von sv3D so allgemein wie möglich gehalten. Abbildung 7.1 zeigt mehrere Container, welche die einzelnen Elemente hervorhebt. In der Ansicht sind die Polyzylinder eine Zeile Text aus der Source-Datei, die mit einem Container assoziiert ist. Ein Container ist eine Source-Datei. Die visuellen Komponenten eines Containers werden von Werten aus der Source-Datei, die selbst bestimmt werden können, geladen. In der Abbildung 7.1 werden die Container gezeigt, in der die Farbe der Polyzylinder zu der Kontrollstruktur und die Höhe der Polyzylinder dem Nesting-Level zugeordnet wird. Der Name der Source-Datei ist an jedem Container angeheftet. In sv3D wird angestrebt, dass Container untereinander Relationen besitzen, um die Darstellung der hierarchischen Struktur der Daten und Beziehungen zu ermöglichen, wie es in ExplorViz bereits möglich ist.

Unterstützung von Benutzerinteraktionen

sv3D erlaubt verschiedene Benutzerinteraktionen wie das Navigieren und Zoomen in der Ansicht. Der Benutzer kann in der Ansicht in jede Richtung navigieren und schwenken. Damit kann der Benutzer das System aus jedem Blickwinkel betrachten. Damit auch große Software von dem Benutzer einfach betrachtet werden kann, ist das Zoomen in verschiedenen Geschwindigkeiten möglich. Jeder Container kann in der Ansicht auch einzeln betrachtet werden und Interaktionen sind auf diesen Container übertragbar. Dabei wird nur der einzelnen Container in der Ansicht verändert. Des Weiteren ist es dem Benutzer möglich, einen *Snapshot* von der aktuellen Ansicht zu erstellen, zu speichern und einzusehen. Außerdem ist es möglich, eine Reihe von *Snapshots* zu erstellen und diese als Historie abzuspielen. In ExplorViz dagegen ist das Erstellen einer Historie aus

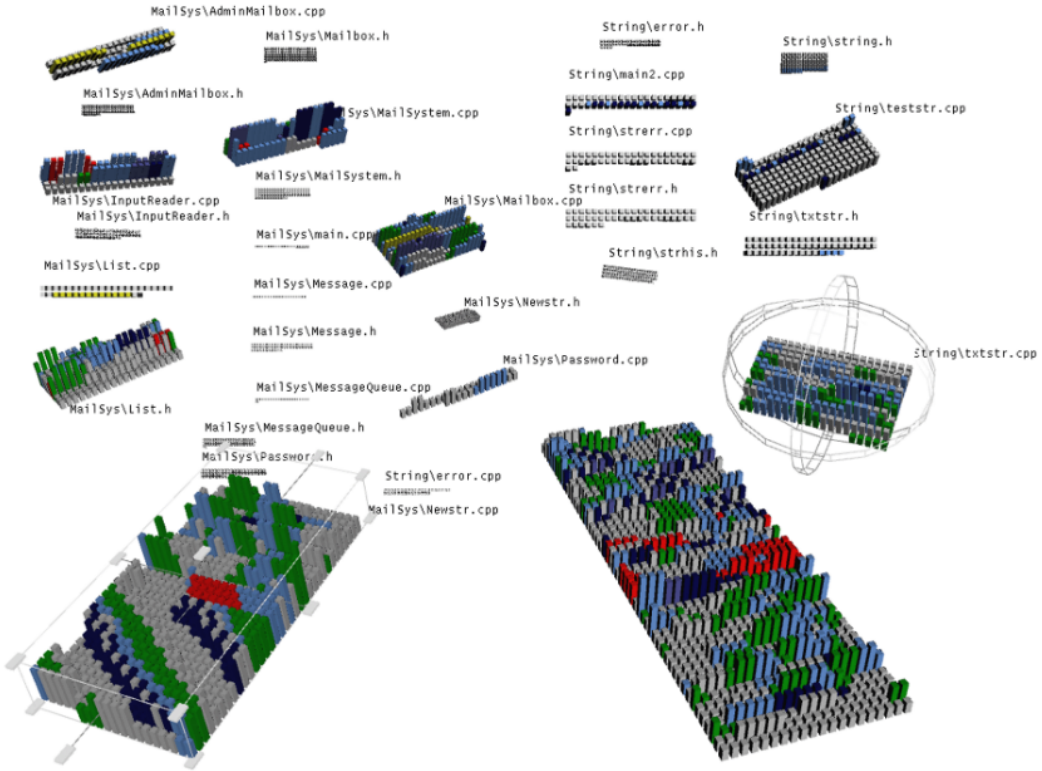


Abbildung 7.1. Überblick einer Software in sv3D. Farben repräsentieren die Kontrollstrukturen und die Höhe der Polyzylinder repräsentieren das Nesting-Level [Marcus u. a. 2003].

den *Snapshots* und das Abspielen nicht möglich. Dafür ist es dem Benutzer in ExplorViz möglich, einen *Snapshot* mit anderen Benutzern zu teilen. Diese Funktion ist in sv3D nicht vorhanden.

7.2. CocoViz

CocoViz [Boccuzzo und Gall 2007] visualisiert Softwaremetriken mithilfe kognitiver Metaphern, um die Verständlichkeit zu verbessern. Dabei kommt ein Software-Metrik-Konfigurator zum Einsatz, der Metrikkombinationen und Layouts ermöglicht.
Metrik-Cluster gruppieren spezifische Metriken, um Verständlichkeit der Software zu erhöhen. Ein Beispiel ist die Kombination von Struktur- und Komplexitätsmetriken, um komplexe Softwarekomponenten zu identifizieren. Metrik-Clusters werden dem Metrik-Konfigurator bereitgestellt. Der Metrik-Konfigurator funktioniert ähnlich wie ein Audio-

mixer und ermöglicht die Anpassung von Metriken durch Filter, Normalisierung und Transformation. Dies erleichtert die schnelle Anpassung und Vergleichbarkeit von Visualisierungen über ein Projekt hinweg. Für die Visualisierung sollen Metaphern eine intuitive Darstellung bieten. Es gibt die Hausmetapher, die Tischmetapher und die Speermetapher. Dabei gestalten die Klassen jeweils eine der Metaphern. Eine gut geformte Metapher repräsentiert eine gut gestaltete Klasse, so wie eine schlecht geformte Metapher eine problematische Klasse darstellt. In Abbildung 7.2 sind die drei Metaphern für eine Software

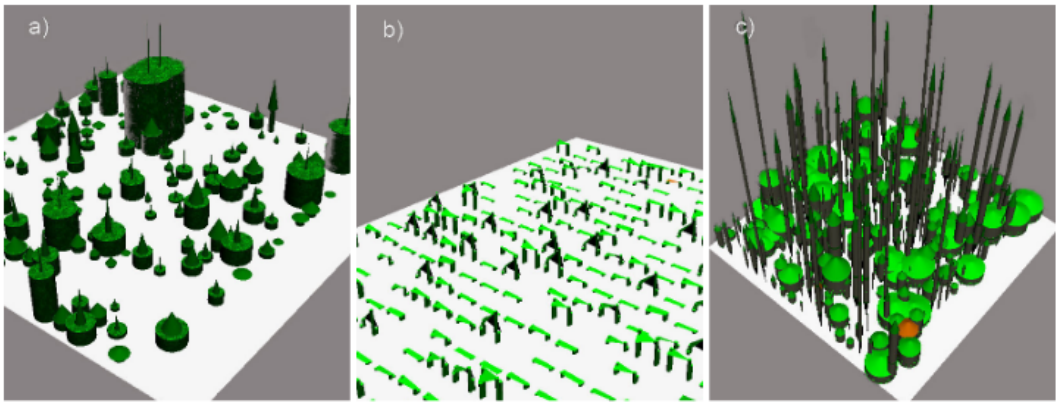


Abbildung 7.2. Die drei verschiedenen Metaphern, um eine Software zu visualisieren. Abbildung a) zeigt die Hausmetapher, b) die Tischmetapher und c) die Speermetapher [Boccuzzo und Gall 2007].

dargestellt. Hierbei fallen schnell die breiten, kleinen Metaphern und die schmalen, länglichen Metaphern auf. Diese Formen sollen problematische Klassen repräsentieren.
In CocoViz wurde zudem ein Konzept für das Bewahren von Visualisierungen implementiert. Dabei ist es dem Benutzer möglich, interessante Metaphern zu markieren und diese dann in der Visualisierung zu bewahren. Dies ist dazu gedacht, die interessanten Klassen später mit anderen Entwicklern zu teilen.
CocoViz hat einen ganz anderen Ansatz als ExplorViz und ermöglicht dabei ebenfalls das Speichern von Visualisierungen. Die Idee für das Speichern eines Visualisierungsstands ist dieselbe. Auch in CocoViz wird über das Teilen gesprochen, wird aber nicht weiter spezifiziert. Es ist zu bedenken, dass hierbei möglicherweise nur das Zeigen der eigenen Softwarevisualisierung gemeint ist und nicht, wie der Ansatz in ExplorViz, das Teilen über eine URL oder ähnliche Möglichkeiten.

Kapitel 8

Fazit und Ausblick

In diesem Kapitel fassen wir die Implementierung für das Persistieren und Teilen einer Ansicht und die Erweiterung der Git-Hosting Anbindung zusammen und geben einen Ausblick in zukünftige Arbeiten.

8.1. Fazit

In dieser Arbeit haben wir für ExplorViz eine Snapshotfunktion implementiert. Dem Benutzer ist es nun möglich, die Ansicht einer Landschaft zu persistieren und diese auch mit anderen Benutzern zu teilen. Da aus solchen Ansichten häufig Fehlerquellen erkannt werden, haben wir außerdem die Git-Hosting Anbindung angepasst. Es ist nun möglich, in einem Issue einen *Snapshot* anzulegen. Die Erstellung eines Issues wurde zudem vereinfacht.
Um die Funktionalität und die Benutzerfreundlichkeit zu testen, wurde eine Evaluation mit 14 Probanden durchgeführt. In der Evaluation wurde die Benutzerfreundlichkeit und die generelle Funktionsweise der Implementierung evaluiert. Die Resultate der Evaluation waren als sehr positiv zu vermerken und die Probanden haben die neuen Erweiterungen als benutzerfreundlich eingestuft. Außerdem erhielten wir viel detailliertes Feedback, anhand dessen die Funktionalität und die Benutzerfreundlichkeit noch verbessert werden kann. Die Funktion des Speicherns einer Ansicht wurde als sehr hilfreich empfunden und sollte in Zukunft erweitert werden.

8.2. Ausblick

In diesem Abschnitt werden weitere Ideen für zukünftige Erweiterungen präsentiert, die auch aus den Resultaten der Evaluation hervorgehen.

Erweiterung der Snapshotfunktion

Einen *Snapshot* als kollaborativen *Snapshot* anzubieten, wäre eine sinnvolle Möglichkeit, die Snapshotfunktion zu erweitern. Hier kann ein Benutzer, wie bisher möglich, einen *Snapshot* erstellen und diesen mit anderen Benutzern teilen. In diesem *Snapshot* werden dann fortlaufend und automatisch alle hinzugefügten Änderungen gespeichert. Damit speichert

der *Snapshot* einen fortlaufenden Prozess und eine Historie wäre umsetzbar, wie es in der verwandten Arbeit [Marcus u. a. 2003] beschrieben ist. Außerdem werden durch andere Benutzer getätigte Änderungen des *Snapshots* automatisch gespeichert. Die Änderungen werden benutzerspezifisch dokumentiert, sodass eine Nachverfolgung möglich ist.
Als nicht intuitiv wurden die Erstellung und der Export eines *Snapshot* eingestuft. Für Verwirrung dabei sorgte, dass ein *Snapshot* nur gespeichert oder exportiert werden kann. Hier ist ein eigener Button mit dem Namen *Create and Export* denkbar. Außerdem könnte die Startseite modifiziert werden. Anstatt einer lange Startseite wäre eine Navigationsleiste denkbar, bei der in Landschaften und *Snapshots* unterteilt wird.

Erweiterung der Git-Hosting Anbindung

Das *Restructuring*-Menü in ExplorViz stellt sich als für einen Benutzer nicht intuitiv heraus. In dieser Arbeit wurde nur mit der Git-Hosting Anbindung gearbeitet und das Menü dabei gleich belassen. Ein weiteres Menü für die Erstellung eines Issues ist aus Benutzersicht sinnvoll. Diese Anmerkung wurde auch in der Evaluation vermerkt. Dadurch wird das *Restructuring*-Menü abgekapselt und es entsteht eine visuelle Trennung des *Restructuring*-Menüs von dem Erstellen eines Issues. Der Gesamteindruck wird dadurch für den Benutzer übersichtlicher. Einen Issue in ExplorViz zu erstellen, sollte unabhängig von der Umstrukturierung in ExplorViz möglich sein.

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)

[Boccuzzo und Gall 2007] S. Boccuzzo und H. Gall. CocoViz: Towards Cognitive Software Visualizations. In: *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. 2007, Seiten 72–79. DOI: 10.1109/VISSOF.2007.4290703. (Siehe Seiten 44, 45)

[Ember.js Autoren 2024] T. I. Ember.js Autoren. *Anatomy of an Ember App*. https://guides.emberjs.com/v5.11.0/getting-started/anatomy-of-an-ember-app/. Abrufdatum: 06. September 2024. 2024. (Siehe Seite 7)

[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, 8)

[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 5)

[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)*. Sep. 2022, Seiten 182–187. DOI: 10.1109/IC2E55432.2022.00026. (Siehe Seiten 8, 10)

[Marcus u. a. 2003] A. Marcus, L. Feng und J. I. Maletic. 3D representations for software visualization. In: *Proceedings of the 2003 ACM Symposium on Software Visualization*. SoftVis '03. San Diego, California: Association for Computing Machinery, 2003, 27–ff. DOI: 10.1145/774833.774837. (Siehe Seiten 43, 44, 48)

[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 6)

[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)

[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 Seite 1)

[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 2024] P. Wulff. Kollaborative Annotationen und die Persistierung durch Snapshots in einer 3D Software Visualisierung. Bachelorarbeit. Christian-Albrechts-Universität zu Kiel, 2024. (Siehe Seite 16)

[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 Seite 36)

[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)