Backend-Entwicklung eines kollaborativen Tools zur Definition von Design Challenges

Studienarbeit II

des Studienganges Elektrotechnik
an der Dualen Hochschule Baden-Württemberg Mannheim

von
Johannes Schackniß

27.06.2021

Bearbeitungszeitraum:	12.04.2021 - 27.06.2021
Matrikelnummer, Kurs:	4175962, TEL18AAT
Ausbildungsfirma:	ABB Automation GmbH
Betreuer der Dualen Hochschule:	Dipl.-Inf. Reiner Hüchting

Abstract

Diese Studienarbeit umfasst die Backend-Entwicklung eines kollaborativen Tools zur Definition von Design Challenges. Eine Design Challenge ist die Grundlage für Design Thinking. Design Thinking umfasst Methoden und Technik, um Teams beim Lösen von Problemen und Entwickeln von Innovationen zu unterstützen. Das Tool soll Benutzer bei der Definition einer optimalen Design Challenge unterstützen. Dies ist in Form eines Anwendungsfalls thematisiert. Das Projekt ist mit der Programmiersprache Go umgesetzt. Zur Datenspeicherung wird die dokumentenorientierte Datenbank MongoDB eingesetzt. Um den Prozess sowohl in Go, als auch in der Datenbank abzubilden, wurden zwei Datenmodelle entwickelt. Des Weiteren sind CRUD-Operationen und dazugehörige Unit-Tests zur Umsetzung des Anwendungsfalls implementiert. Mit Hilfe des WebSocket-Protokolls und des „websocket“-Paketes ist die Möglichkeit zur kollaborativen Zusammenarbeit, sowie eine API umgesetzt. Clients können Funktionen über die API anfragen und die Antwort wird anschließend an alle Clients gesendet. Sowohl Anfrage, als auch Antwort sind als JSON formatiert.

This project thesis covers the backend development of a collaborative tool for defining *design challenges*. A *design challenge* is the foundation for *design thinking*. *Design thinking* includes methods and techniques to help teams solve problems and develop innovations. The tool is designed to help users define an optimal *design challenge*. This is presented in the form of a use case. The project is implemented using the Go programming language. The document-oriented database MongoDB is used for data storage. In order to map the process in Go as well as in the database, two data models were developed. Furthermore, CRUD operations and corresponding unit tests are implemented to fulfil the use case. With the help of the WebSocket protocol and the “websocket” package, the feature to collaborate and an API are implemented. Clients can request functions through the API and the response is then sent to all clients. Both request and response are formatted as JSON.

Erklärung

Ich, Johannes Schackniß, versichere hiermit, dass ich meine Projektarbeit mit dem Thema: „Backend-Entwicklung eines kollaborativen Tools zur Definition von Design Challenges“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.
Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Mannheim, 27.06.2021

Ort, Datum

Unterschrift (Johannes Schackniß)

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
CRUD	Create Read Update Delete
DBMS	Datenbankmanagementsystem
gRPC	gRPC Remote Procedure Calls
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
JSON	JavaScript Object Notation
NoSQL	Not only Structured Query Language
RFC	Request for Comments
RPC	Remote Procedure Calls
SQL	Structured Query Language
TCP	Transmission Control Protocol
UML	Unified Modeling Language

1 Einleitung

„Design Thinking bezeichnet eine Sammlung von agilen Methoden, Tools und Techniken, die – in Kombination – Teams unterstützen, komplexe Probleme zu lösen und Innovationen zu schaffen. Ziel ist es, nutzerzentrierte Ideen zu entwickeln, zu testen und somit den größtmöglichen Wert für Unternehmen, Stakeholder und Nutzer zu generieren.“[1, S. 17] Die Sammlung von agilen Methoden, Tools und Techniken ist in Form des Design-Thinking-Prozesses abgebildet. Dieser besteht aus mehreren Phasen. [1, S. 17, 28–31]

Bevor ein Team den Design-Thinking-Prozess beginnen kann, ist eine Formulierung der Aufgabenstellung bzw. Problemhypothese, auch Design Challenge genannt, erforderlich. „Nur wenn die Challenge klar formuliert wurde, kann der eigentliche Design-Thinking-Prozess beginnen.“[1, S. 35] Eine gute Design Challenge sollte weder zu geschlossen, noch zu offen formuliert sein, um den Design-Thinking-Prozess erfolgreich durchzuführen. Dies stellt meist die erste Herausforderung beim Design Thinking dar. [1, S. 33 ff.], [2, S. 16]

Der Aufgabe, den Prozess zur Formulierung einer Design Challenge einfach und verständlich zu gestalten, stellten sich Maximilian Kürschner und ich im Rahmen des Hackathons „Mesh Innovation Summit“. Bei dem Hackathon entwickelten wir ein Mockup einer Webapplikation, dass ein Team unterstützt die Theorie in die Praxis umzusetzen. Dabei wird das Team oder ein einzelner Benutzer Schritt für Schritt durch den Prozess zur Definition einer Design Challenge geführt. In der finalen Runde des Hackathons erhielt die Idee viel Zuspruch durch andere Teilnehmer und der Jury, sodass wir uns zur Weiterentwicklung des Projektes entschieden. [3]

Die Entwicklung des soeben beschriebenen kollaborativen Tools zur Definition von Design Challenges ist Thema dieser Arbeit. Dabei liegt der Fokus auf der serverseitigen Entwicklung, dem Backend.

2 Aufgabenstellung

Die Studienarbeit thematisiert die Backend-Entwicklung eines kollaborativen Tools zur Definition von Design Challenges und ist im Zeitraum von 10 Wochen umzusetzen. Die Evaluierung und Auswahl einer geeigneten Programmiersprache ist Teil der Studienarbeit. Folgende Funktionalitäten sind bei der Umsetzung zu realisieren:

- persistente Datenspeicherung des Prozesses
- kollaborative Zusammenarbeit
- Application Programming Interface (API)

Die persistente Datenspeicherung des Prozesses umfasst die Entwicklung eines geeigneten Datenmodells, die Evaluierung der Art der Datenspeicherung, sowie die Implementierung. Für die kollaborative Zusammenarbeit ist die Umsetzung zu evaluieren und zu implementieren. Hierbei soll es einem Team möglich sein, den Prozess zur Definition einer Design Challenge gemeinsam in einer Webapplikation durchzuführen. Die API soll Funktionalitäten zum Schreiben und Lesen von Daten bieten, sowie den Prozess Challenge-Formulierung abbilden.

Des Weiteren sind Tests für die implementierten Funktionen zu erstellen. Diese sind bestmöglich zu automatisieren. Das Testen der API soll mit Hilfe eines einfachen Clients, z.B. über ein Terminal, erfolgen. Zudem ist der Code des Backends zu dokumentieren.

3 Verwandte Literatur

Dieses Kapitel der Studienarbeit umfasst einen Überblick der verwendeten Literatur. Somit ist ein schneller Einstieg für den Leser möglich. Des Weiteren beinhaltet das Kapitel weiterführende Literatur, um die Vertiefung nur oberflächlich behandelter Themen zu ermöglichen.

Da eine Design Challenge die Voraussetzung für den Design-Thinking-Prozess ist, bietet sich zunächst die Auseinandersetzung mit dem Thema Design Thinking an. Hier bietet das Buch „Design Thinking: Innovationen effektiv managen“ [1] gute Erklärungen zum Prozess und gibt erste Einblicke zur Definition einer Design Challenge. Zur Vertiefung des Themas Design Challenge und des Prozesses zur Definition, verfügt das Workshopmaterial „Startpunkt: Herausforderung festlegen“ [2] über ausführliche weiterführende Inhalte.

Zur Umsetzung des Projektes kommt die Programmiersprache Go und die Entwicklungsumgebung Visual Studio Code zum Einsatz. Die interaktive Lernumgebung „A Tour of Go“ [4] bietet einen einfachen Einstieg in Go und die grundlegenden Programmierprinzipien. Details zur Programmiersprache Go enthält die Dokumentation [5]. Um die Funktionalität des Codes zu testen, kommen Unit-Tests zum Einsatz. Hierfür enthält [6] eine kurze Anleitung. Weiterführende Informationen zu Unit-Tests in Go sind in der Dokumentation des „testing“-Paketes [7] enthalten. Für die Versionskontrolle des Projektes kommt Git zum Einsatz. Die Dokumentation [8] der Software bietet Anleitungen als Einstieg, sowie detaillierte Informationen zur Software. Um das Projekt zusätzlich zu sichern und eine spätere Zusammenarbeit zu ermöglichen, kommt zudem GitHub zum Einsatz. GitHub ermöglicht eine netzbasierte Versionsverwaltung und verfügt dabei über Projektmanagement-Tools. Anleitungen zur Verwendung von GitHub und allen verfügbaren Funktionen sind in der Dokumentation [9] enthalten.

Die Backend-Entwicklung umfasst unter anderem die Datenspeicherung des Prozesses. Grundlegende Definitionen und Eigenschaften zum Thema Datenbanken sind im Buch „Grundkurs Datenbanksysteme: Von den Konzepten bis zur Anwendungsentwicklung“ [10] zu finden. Weiterführende Informationen zu Not only Structured Query Language (NoSQL)-Datenbanken und dokumentenorientierten Datenbanken bieten das Buch „NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken“ [11] und der Onlineartikel „Informationen zu NoSQL-Datenbanken“ [12]. Mit Hilfe der „Quick Start“-Anleitungen [13], [14], [15] und [16] ist eine grundlegende Nutzung von MongoDB mit Go möglich. Genauere Details zur Nutzung von MongoDB sind der Dokumentation [17] zu entnehmen.

Für die Umsetzung der Anforderung - kollaborative Zusammenarbeit - sind gRPC Remote Procedure Calls (gRPC) und das WebSocket-Protokoll in Betracht gezogen. Allgemeine und tiefgründige Informationen zu gRPC sind in der offiziellen Dokumentation [18] enthalten. Der Onlineartikel „Was ist WebSocket?“ [19] gibt einen allgemeinen Überblick zum WebSocket-Protokoll. Genaue Details dazu sind im Standard Request for Comments (RFC) 6455 [20] der Internet Engineering Task Force (IETF) zu finden. Die Implementierung in Go ist in der Dokumentation des „websocket“-Paketes [21] genauer beschrieben.

4 Vorgehensweise

Zu Beginn des Projektes erfolgt die Projektplanung. Diese umfasst sowohl die Definition der Anforderungen, als auch die zeitliche Planung. Für die gesamte Projektumsetzung, dies umfasst das Projektmanagement und die Softwareentwicklung, kommt GitHub zum Einsatz. Hier können sogenannte Issues, die gewünschte Anforderungen oder gefundene Bugs beinhalten, erstellt werden. Dabei besteht die Möglichkeit erstellte Issues einer Person, einem Label, einem Milestone und einem Projekt zuzuordnen. Somit ist eine zeitliche, kategorische und personelle Zuordnung möglich. Die Anforderungen sind als Issues, wie in Abbildung 4.1 dargestellt, formuliert und die zeitliche Umsetzungsplanung erfolgt durch die Zuordnung von Milestones. Die Projektübersicht ist mit Hilfe eines „Projectboards“ realisiert. [22]

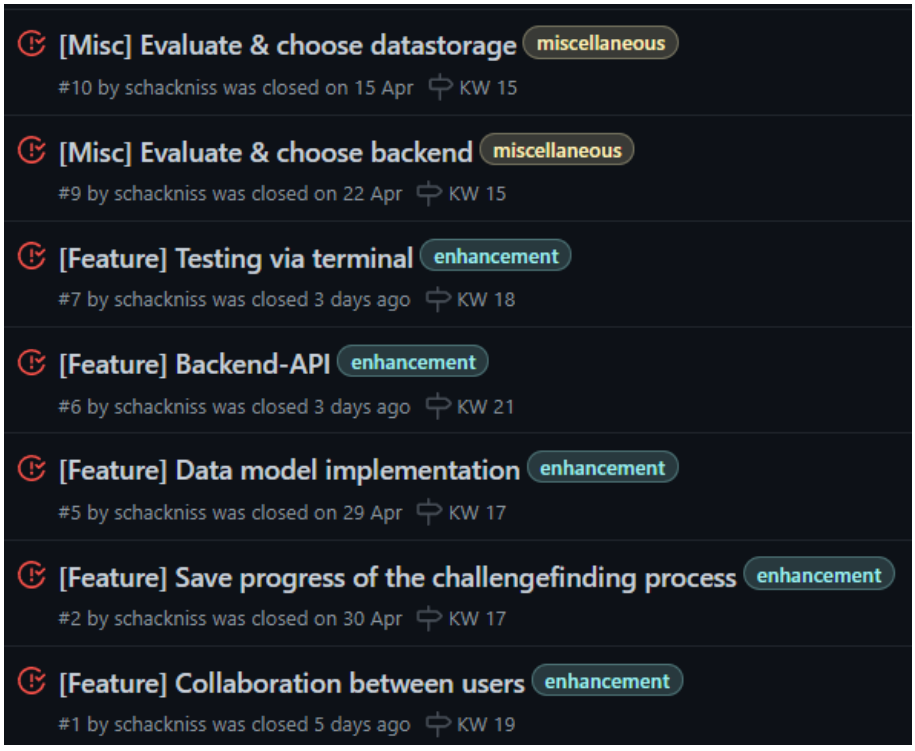

Abbildung 4.1: Auszug der Issues

Nach Abschluss der Anforderungsdefinition und zeitlichen Planung folgt die theoretische Vorbetrachtung bzgl. dem Thema Design Challenge. Hierbei wird der Prozess zur Formulierung und ein möglicher Anwendungsfall des Tools beschrieben.

Daraufhin findet die Evaluation einer geeigneten Programmiersprache und die Beschreibung des Projektaufbaues statt. Die Grundanforderungen aus Kapitel 2 werden einzeln implementiert und durch die API zusammengeführt.

Zunächst findet die Evaluation verschiedener Datenbankformen statt, sowie die Entwicklung von Datenmodellen zur Abbildung des Design Challenge Prozesses. Darauffolgend ist die gewählte Datenbank aufzusetzen und die benötigten Create Read Update Delete (CRUD)-Operationen sind zu implementieren. Um die gewünschte Funktionsweise der implementierten Funktionen zu gewährleisten, sind Tests durchzuführen.

Anschließend folgt die Implementierung der API. Hierbei steht zunächst die Wahl einer Netzwerktechnologie und deren Implementierung im Vordergrund. Zudem ist die API zu definieren und ein Anfragen-Handler zu implementieren.

5 Design Challenge

Dieses Kapitel beinhaltet das Thema Design Challenge im Allgemeinen, sowie den Prozess zur Formulierung und ein Usecase bzw. Anwendungsfall des Tools.

Die Design Challenge gibt den Rahmen für den Design-Thinking-Prozess vor. Somit ist diese von großer Bedeutung und sollte wohldurchdacht formuliert sein. Die Design Challenge sollte spezifisch genug sein und dabei inspirieren. Jedoch sollte sie den Lösungsraum nicht beschränken und somit „lösungsfrei“ formuliert sein. [2, S. 12]

5.1 Prozess zur Definition

Grundlegend gibt es verschiedene Methoden zur Formulierung einer Design Challenge. Bei allen Methoden ist die Neu- oder Redefinition der Design Challenge jederzeit möglich. [2, S. 12]

Zunächst müssen die Anwender ein Thema wählen, dass im Design-Thinking-Prozess bearbeitet werden soll. Anschließend wird das Thema genauer betrachtet, um eine mögliche Challenge besser erfassen zu können.

Hierbei kann z.B. eine Mindmap, wie in Abbildung 5.1 zu sehen, zum Einsatz kommen. Die Mindmap unterstützt das Team dabei, die Relevanz des Themas besser einzuschätzen. Dabei werden Ideen zu den Bedürfnissen, Interessen und der Machbarkeit in Bezug auf das Thema zusammengestellt. [2, S. 13]

Eine weitere Methode, um das Thema besser zu erfassen, ist das Zusammenstellen möglicher Handlungsorte, Personen bzw. Charaktere und potentieller Probleme mit Bezug zum gewählten Thema. In diesem Schritt der Formulierung, wie in Abbil-

dung 5.2 dargestellt, sind alle Einfälle von äußerst hohem Wert. Somit ist es möglich das Thema möglichst diversifiziert und detailliert zu erfassen. [2, S. 14]

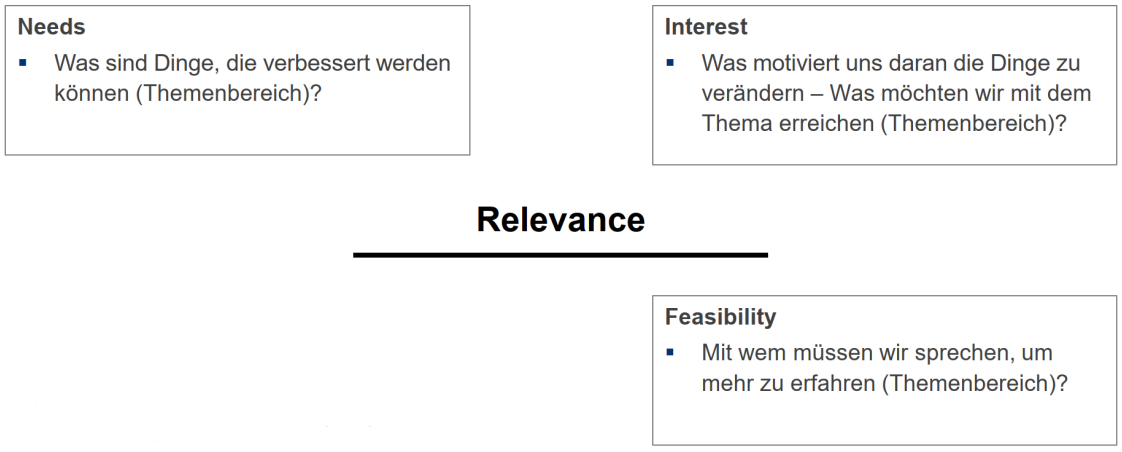

Abbildung 5.1: Tool 1: Mindmap zur Formulierung einer geeigneten Challenge [2, S. 13]

In der Praxis hat sich die zweite Methode, wie in Abbildung 5.2 zu sehen, zur genaueren Erfassung des Themas als nützlich erwiesen. Somit wird diese Methode als Referenz im weiteren Verlauf der Studienarbeit genutzt.

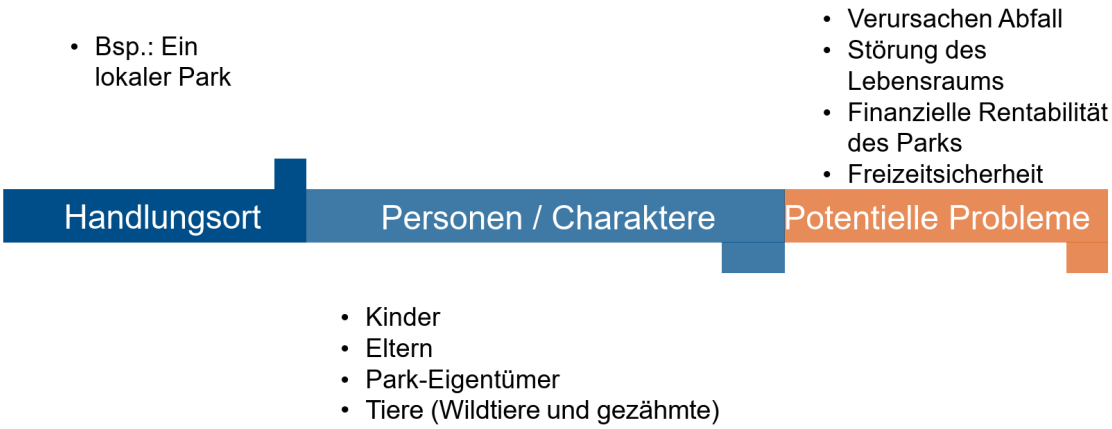

Abbildung 5.2: Tool 2: Möglichkeit die Herausforderung zu erfassen [2, S. 14]

Nach der Ideensammlung bzgl. des Themas folgt die Fokus-Phase. Hierbei muss sich das Team nun fokussieren und sich für je einen Handlungsort, Charakter und ein potentielles Problem entscheiden. Dies dient zur Eingrenzung, sodass eine konkrete Formulierung möglich ist.

Anschließend folgt die Formulierung der Design Challenge. Hierbei ist eine Formulierung nach dem folgenden Schema am einfachsten: „Wie können wir [Personen] im

[Handlungsort] mit dem [potentiellen Problem] helfen?“. Dies ist die erste Version der Design Challenge. [2, S. 15]

Um sicherzustellen, dass die Formulierung weder zu offen, noch zu eingegrenzt ist, folgt der Checkup. Dies erfolgt wieder anhand der drei Kategorien Handlungsort, Charaktere und potentielle Probleme. Zu jeder Kategorie muss sich das Team die folgenden zwei Fragen stellen [2, S. 16]:

1. Gibt es in dieser Situation mehrere Handlungsorte / Charaktere / potentielle Probleme? [2, S. 16]
2. Gibt es in dieser Situation Gemeinsamkeiten zwischen den Handlungsorten / Charakteren / potentiellen Problemen? [2, S. 16]

Falls beide Fragen für jede Kategorie mit „ja“ beantwortet werden, so handelt es sich um eine sehr gute Formulierung. Bei einer solchen Design Challenge ist der Rahmen für den Lösungsraum weder zu offen bzw. breit, noch zu eingegrenzt. Wenn die erste Frage in einer oder in mehreren Kategorien mit „nein“ beantwortet wird, so ist die Challenge zu eng gefasst. Hier sollte das Team zurück zur Fokus-Phase kehren und ab diesem Punkt erneut starten. Dabei ist es wichtig, dass das Team die Design Challenge nun offener bzw. breiter fasst. Falls hingegen die zweite Frage mit „nein“ beantwortet wird, so ist die Frage zu offen formuliert und der Lösungsraum ist zu breit. Auch in dieser Situation ist ein Neustart des Prozesses ab der Fokus-Phase sinnvoll, sodass der Rahmen der Design Challenge weiter eingegrenzt wird. Der soeben beschriebene Prozess zur Formulierung bzw. Definition einer Design Challenge ist nochmals grafisch in Abbildung 5.3 dargestellt. [2, S. 16]

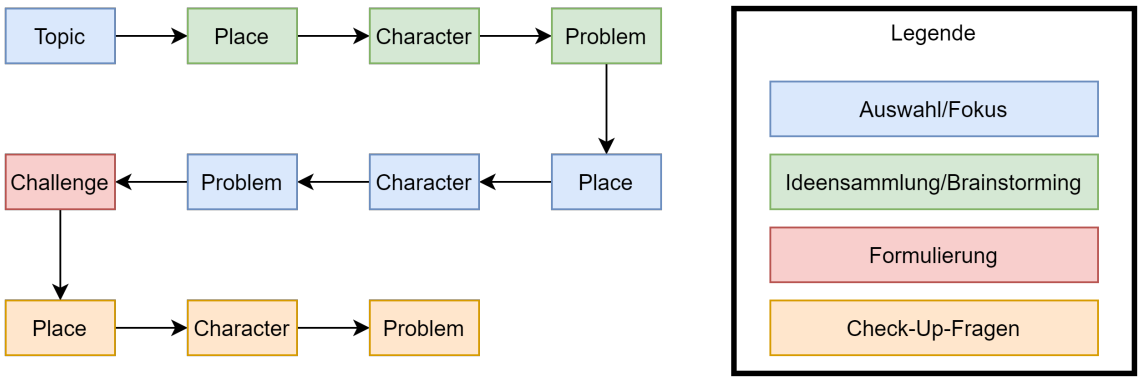

Abbildung 5.3: Prozess zur Formulierung einer Design Challenge

5.2 Usecase/Anwendungsfall

Wie bereits in der Einleitung, in Kapitel 1, beschrieben, wurde im Rahmen eines Hackathons [3] bereits ein Anwendungsfall und eine mögliche grafische Oberfläche realisiert. An der bereits realisierten grafischen Oberfläche sind auch die Wireframes in Abbildung 5.4 und Abbildung 5.5 orientiert.

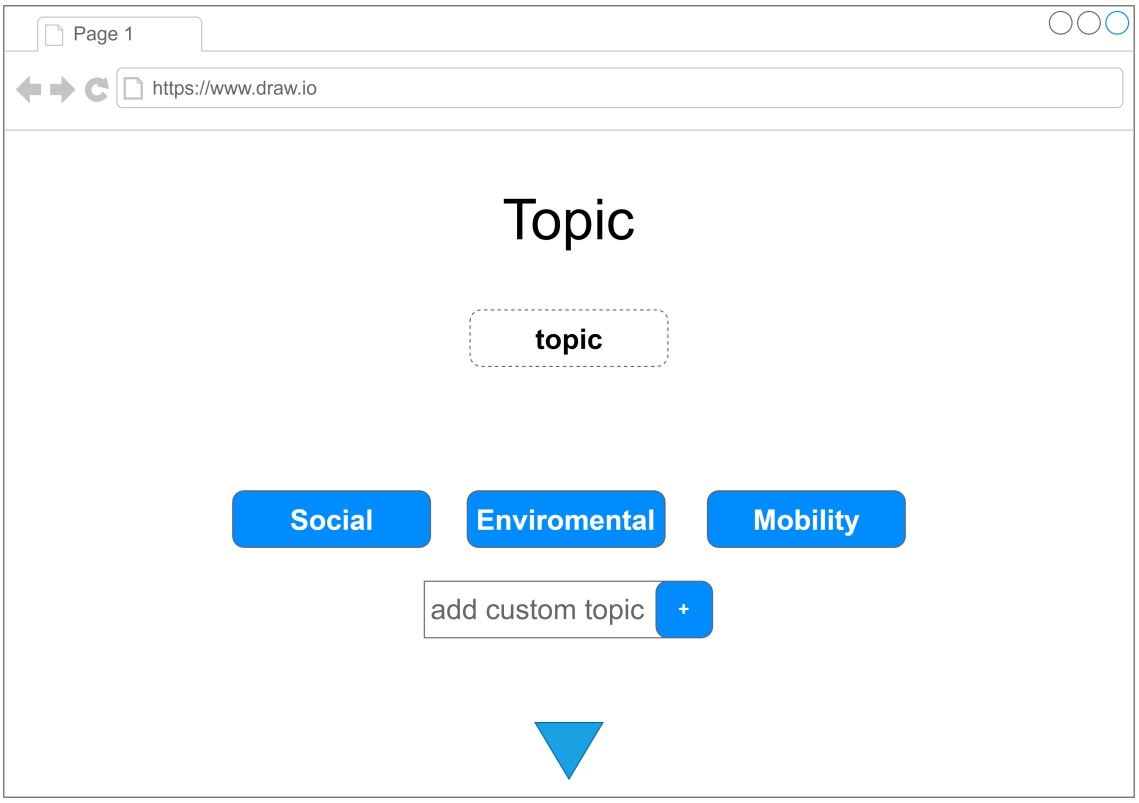

Abbildung 5.4: Wireframe für die Themenauswahl

Das Tool zur Definition einer Design Challenge soll einen einzelnen Benutzer oder ein Team durch den zuvor in Abschnitt 5.1 beschriebenen Prozess führen. Dabei werden die einzelnen Schritte linear durchlaufen. Die Iteration von Schritten ist in diesem Anwendungsfall nicht vorgesehen, da dies die Komplexität zu sehr erhöht. Der Benutzer soll in der jeweiligen Ansicht nur die aktuelle Phase (z.B. Fokus-Phase) und dazugehörige Kategorie (z.B. Handlungsort) sehen. Um den Prozess zu erleichtern stehen dem Benutzer dabei sowohl bei der Themenauswahl, wie in Abbildung 5.4 dargestellt, als auch bei der Ideensammlung Vorschläge zur Verfügung. Falls der Benutzer bzw. das Team eigene Ideen einbringen möchte, ist dies über eine Textbox möglich.

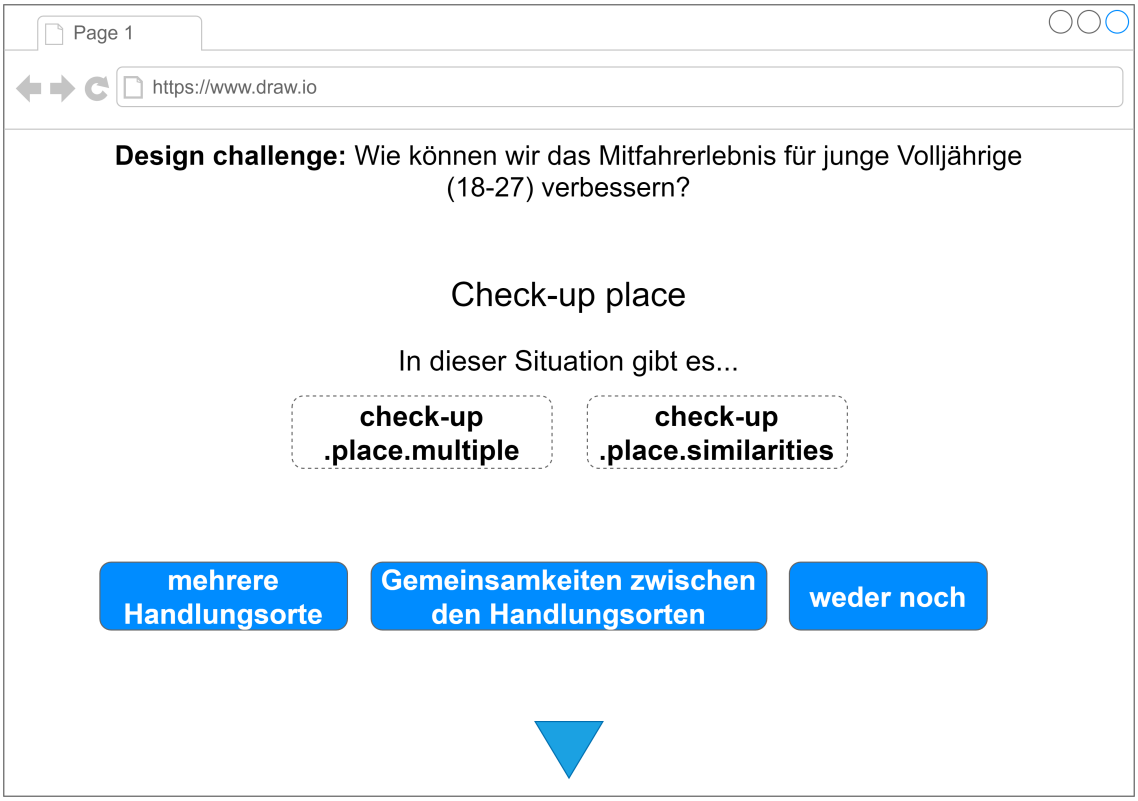

Abbildung 5.5: Wireframe für die Checkup-Fragen zum Handlungsort

Bei den Checkup-Fragen stehen dem Benutzer, wie in Abbildung 5.5 zu sehen, lediglich drei Buttons zur Verfügung, um die Frage zu beantworten. Alle Eingaben durch den Benutzer bzw. das Team werden gespeichert, sodass das Team den Prozess jederzeit pausieren kann. Dies ermöglicht es auch, den Anwendungsfall zukünftig durch die Iteration von Schritten zu erweitern. Zudem sieht das Usecase vor, dass alle Beteiligten die Eingabe der anderen Nutzer in Echtzeit dargestellt bekommen. Somit ist die kollaborative Zusammenarbeit gewährleistet.

6 Programmiersprache

Dieses Kapitel der Studienarbeit umfasst sowohl die Evaluation und Auswahl einer geeigneten Programmiersprache, zur Umsetzung der Anforderungen, als auch den Aufbau des realisierten Go-Projektes.

6.1 Evaluation der Programmiersprache

Das Backend stellt die gesamte Logik des Tools dar. Somit ist die Wahl der Programmiersprache von großer Bedeutung, um die benötigten Anforderungen umzusetzen. Zu Beginn der Evaluation standen die Programmiersprachen PHP, JavaScript und Go zur Auswahl. Da die Programmiersprache PHP auf Basis von Erfahrungen aus der Vorlesung „Web Programmierung“ frühzeitig ausgeschlossen wurde, liegt in den folgenden Abschnitten der Fokus auf den Sprachen JavaScript und Go.

JavaScript ist eine „objektorientierte Skriptsprache, die Mitte der 1990er-Jahre von der Firma Netscape entwickelt wurde, um Webseiten dynamisch gestalten zu können (HTML).“[23] Der Code wird dabei nicht kompiliert, sondern vom Browser interpretiert und ausgeführt [23]. Eine weitere Eigenschaft von JavaScript ist die dynamische Typisierung, sodass Variablen keine typenbasierten Einschränkungen haben. Werte bekommen einen Datentyp, als Eigenschaft zur aktuellen Laufzeit, zugewiesen. Somit ist eine Laufzeitumgebung notwendig, um JavaScript als Programmiersprache für ein Backend zu nutzen. Hier kommt in der Regel die asynchrone, ereignisgesteuerte Laufzeitumgebung NodeJS zum Einsatz [24]. Mit JavaScript und NodeJS ist kein Multi-Threading und somit auch keine Nebenläufigkeit möglich [24]. Jedoch haben JavaScript und NodeJS eine große Benutzergruppe, sodass eine Vielzahl von Anleitungen und Bibliotheken zur Verfügung stehen [25].

Go ist eine Open-Source Mehrzweck-Programmiersprache, die 2007 von einem Google Team entwickelt wurde. Es handel sich um eine kompilierbare Sprache mit statischer Typisierung, die Nebenläufigkeit und das Mixin-Konzept unterstützt. Syntaktisch ist Go unter anderem an den Programmiersprachen C und C++ orientiert. Go verfügt über ein integriertes Test-Framework, sowie ein Tool zum Generieren von Code-Dokumentationen. Aufgrund der einfachen Dokumentation stehen unter `https://go.dev/` die Dokumentationen öffentlichen Repositories zentral zur Verfügung. Des Weiteren ist die Nutzung von Bibliotheken sehr einfach gestaltet. Dabei muss lediglich der Link zum Repository der Bibliothek eingebunden werden. [25], [5], [26]

Grundsätzlich haben beide Programmiersprachen Vor- und Nachteile. JavaScript hat bei der Verfügbarkeit von Bibliotheken und Anleitungen einen Vorsprung gegenüber von Go. Hingegen sind Go-Anwendungen performanter und lassen sich aufgrund der Nebenläufigkeit besser skalieren. Jedoch ist JavaScript für Entwickler deutlich komfortabler, da nur eine Programmiersprache für Frontend und Backend zum Einsatz kommt. Dies birgt zugleich die Gefahr, dass keine klare Trennung von Frontend und Backend stattfindet. Die dynamische Typisierung von JavaScript führt dazu, dass Fehler erst zur Laufzeit erkennbar sind. Dank der statischen Typisierung in Go sind Fehler schon beim Kompilieren erkennbar, sodass diese nicht zur Laufzeit geprüft werden müssen. Aufgrund der besseren Performanz, der statischen Typisierung und der syntaktischen Anlehnung an C++ wird für die Backend-Entwicklung die Programmiersprache Go genutzt.

6.2 Aufbau des Go-Projektes

Das Go-Projekt ist zur Versionsverwaltung als Git Repository aufgesetzt und wird als Backup und für die zukünftige Zusammenarbeit in ein Remote-Repository auf GitHub hochgeladen.

Grundsätzlich ist Go-Code in Paketen gruppiert, die Pakete sind wiederum in Module gruppiert. Ein Modul beinhaltet die zum Ausführen notwendigen Abhängigkeiten bzw. Pakete inklusive der Version, sowie die verwendete Go-Version. In einem Modul kann sich immer nur ein Paket pro Verzeichnis befinden. Dabei ist es Konvention, dass der Verzeichnisname gleich dem Paket- bzw. Modulnamen ist. [27], [28]

Der Aufbau des Go-Projektes für das Backend ist vereinfacht in Abbildung 6.1 zu sehen. In der Abbildung stellen Ovale Ordner bzw. Verzeichnisse dar. Die davon ausgehenden Pfeile zeigen auf den jeweils enthaltenen Inhalt. Rote Ovale stehen dabei für Module und blaue Ovale für Packages bzw. Pakete. Dateien sind hingegen als Rechtecke abgebildet.

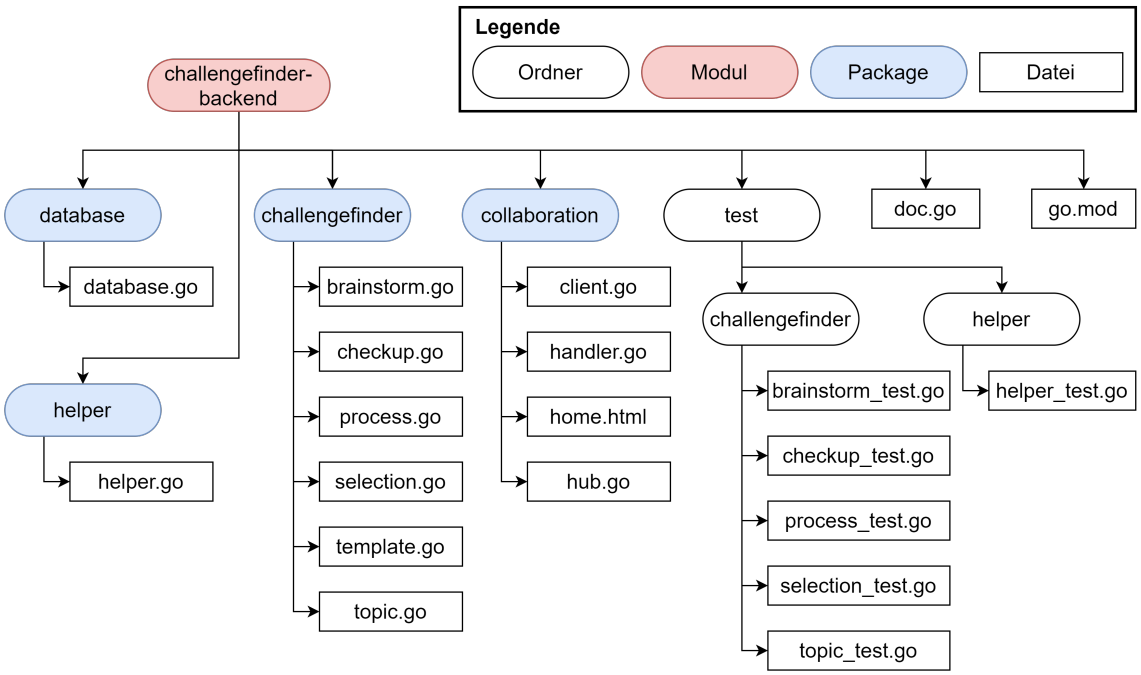

Abbildung 6.1: Aufbau des Go-Projektes

Das gesamte Projekt befindet sich im Verzeichnis „challengefinder-backend“, welches zugleich auch das gleichnamige Modul darstellt. Die Abhängigkeiten und Versionen des Moduls befinden sich in der Datei „go.mod“. In der Datei „doc.go“ ist eine allgemeine Dokumentation des Moduls. Des Weiteren sind die Verzeichnisse „challengefinder“, „collaboration“, „database“ und „helper“, welche zu gleich Pakete sind, Teil des Moduls. Das Paket „challengefinder“ dient der Abbildung des Prozesses zur Definition einer Design Challenge und stellt sowohl CRUD-Operationen zum Speichern des Prozesses, als auch Funktionen zum Setzen von Werten. Im Paket „collaboration“ ist die Anforderung der kollaborativen Zusammenarbeit mit Hilfe von WebSockets, sowie der Anfragen-Handler der API implementiert. Des Weiteren enthält das Paket in der Datei „home.html“ einen einfachen WebSocket-Client zum Senden von Nachrichten. Das „database“-Paket ist für Funktionen vorgesehen, die ausschließlich mit der Datenbank arbeiten. Dieses Paket entstand im Laufe des Lernprozesses mit Go und MongoDB. Für Hilfsfunktionen steht das Paket „helper“ zur Verfügung. Zudem beinhaltet das Modul noch das Verzeichnis „test“, welches Tests der jeweiligen Pakete enthält. [29]

7 Datenspeicherung

Eine grundlegende Anforderung der Themenstellung, Backend-Entwicklung eines kollaborativen Tools zur Definition von Design Challenges, ist die persistente Datenspeicherung des Formulierungs-Prozesses. Dieses Kapitel der Studienarbeit umfasst zunächst einen Vergleich verschiedener Datenbankformen, um eine geeignete Datenbank für den Anwendungsfall zu wählen. Anschließend erfolgt die Modellierung eines geeigneten Datenmodells, um den Prozess sowohl in der verwendeten Datenbank, als auch mit Go abzubilden. Daraufhin ist die Implementierung von CRUD-Operationen, zur Umsetzung des in Abschnitt 5.2 beschriebenen Usecase, thematisiert. Abschließend folgt das Testen der implementierten Funktionen, um deren Funktionalität zu prüfen.

7.1 Datenbankformen

„Eine Datenbank ist eine Sammlung von Daten, die von einem Datenbankmanagementsystem (DBMS) verwaltet wird."[10, S. 3] Das DBMS legt die Struktur der Daten und deren Beziehungen zueinander durch ein Datenbankmodell fest. Hier kommen für die Umsetzung des Backends ein relationales oder ein dokumentenorientiertes Datenbankmodell in die nähere Auswahl. Datenbankmodelle sind häufig auch in Structured Query Language (SQL) und NoSQL kategorisiert. Als SQL-Datenbanken werden Datenbanken mit einem relationalen Modell eingeordnet. Hingegen verfolgen NoSQL-Datenbanken einen nicht-relationalen Ansatz, wie z.B. dokumentenorientierte Datenbanken. [11, S. 2]

„In einer relationalen Datenbank werden alle Informationen in Tabellen abgelegt."[10, S. 21] Die Tabellen unterliegen einem zuvor definiertem Schema und sind in Spalten, sowie Zeilen strukturiert. „Eine Tabelle besteht aus mindestens einer Spalte

und einer endlichen Anzahl von Zeilen. Die Anzahl der Spalten ist ebenfalls endlich. Der Name der Tabelle legt den Informationstyp fest und muss innerhalb der Datenbank eindeutig sein.“[30, S. 16] Jede Spalte hat eine Überschrift, die innerhalb der Tabelle eindeutig ist und den Inhalt bestimmt. Der Inhalt einer Spalte ist immer vom selben Typ. Zeilen enthalten inhaltlich zusammenhängende Daten, die passend zur Reihenfolge der Spalten angeordnet sind. Beziehungen zwischen Tabellen sind bei relationalen Datenbanken verweisbasiert umgesetzt. Die Skalierung von herkömmlichen relationalen Datenbanken ist nur vertikal möglich, sodass die Leistung des einzigen Servers bei erhöhten Anforderungen verbessert werden muss. Bekannte Beispiele relationaler Datenbanken sind MySQL und MariaDB. [10, S. 21], [30, S. 15 f.], [12]

In einer dokumentenorientierten Datenbank sind alle Daten in Dokumenten abgelegt. Diese unterliegen keinem Schema und sind somit sehr flexibel einsetzbar. „Das Schema wird mit Einfügen eines Dokuments zur Laufzeit erzeugt.“[12] In dokumentenorientierten Datenbanken gibt es zwei Beziehungstypen. Zum Einen verweisbasierte Beziehungen zwischen Dokumenten, und zum Anderen eingebettete Beziehungen innerhalb eines Dokuments. Dokumentenorientierte Datenbanken sind sowohl vertikal, als auch horizontal skalierbar. Aufgrund der horizontalen Skalierbarkeit ist das Hinzufügen von zusätzlichen Servern möglich. „Die Datenbank verteilt die Daten bei Bedarf automatisch neu.“[12] Ein bekanntes Beispiel einer dokumentenorientierten Datenbank ist MongoDB. [11, S. 155], [12]

Beide Datenbankmodelle sind zur Speicherung des Prozesses bei der Definition einer Design Challenge geeignet. Jedoch bietet eine dokumentenorientierte Datenbank zum Entwicklungszeitpunkt deutlich mehr Vorteile. Insbesondere das dynamische Schema dieser Datenbanken ist vorteilhaft, da sich dieses mit hoher Wahrscheinlichkeit im Laufe des Entwicklungsprozesses ändert und somit eine agile Entwicklung ermöglicht. Des Weiteren sind bei relationalen Datenbanken lediglich verweisbasierte Beziehungen umsetzbar. Bei dokumentenorientierten Datenbanken hingegen sind verweisbasierte und eingebettete Beziehungen realisierbar. Zudem lassen sich dokumentenorientierte Datenbanken, im Gegensatz zu relationalen Datenbanken, einfacher skalieren. Da eine dokumentenorientierte Datenbank deutlich mehr Vorteile für den vorgesehenen Einsatzzweck gegenüber einer relationalen Datenbank hat, kommt MongoDB zum Einsatz. [10, S. 21], [11, S. 155], [30, S. 15 f.], [12]

7.2 Datenmodell

Nach der Auswahl von MongoDB zur Datenspeicherung beschreibt dieses Kapitel das Datenmodell. Dieses soll sowohl den zuvor in Abschnitt 5.1 beschriebenen Prozess, als auch das in Abschnitt 5.2 beschriebene Usecase abbilden.

Aus dem Usecase geht hervor, dass der Benutzer Daten eingibt und das Tool Vorschläge anzeigt. Somit ist ein Modell für die Benutzereingaben zum Prozess und ein Modell zur Abbildung der Vorschläge bzw. Vorlagen notwendig. MongoDB bietet die Möglichkeit die Dokumente einer Datenbank in Collections zu gliedern. Somit ist im späteren Verlauf der Entwicklung eine einfachere Unterscheidung zwischen Benutzereingaben und Vorlagen möglich. Diese Gliederung wurde auch beim Erstellen der Datenbank, wie in Abbildung 7.1 dargestellt, durchgeführt. Die Datenbank für das Tool trägt den Namen „challenge-finder“ und ist in zwei Collections gegliedert. Die Collection „template“ enthält Dokumente mit Vorschlägen bzw. Vorlagen. Die Benutzereingaben sind in Dokumenten der Collection „process“ gespeichert.

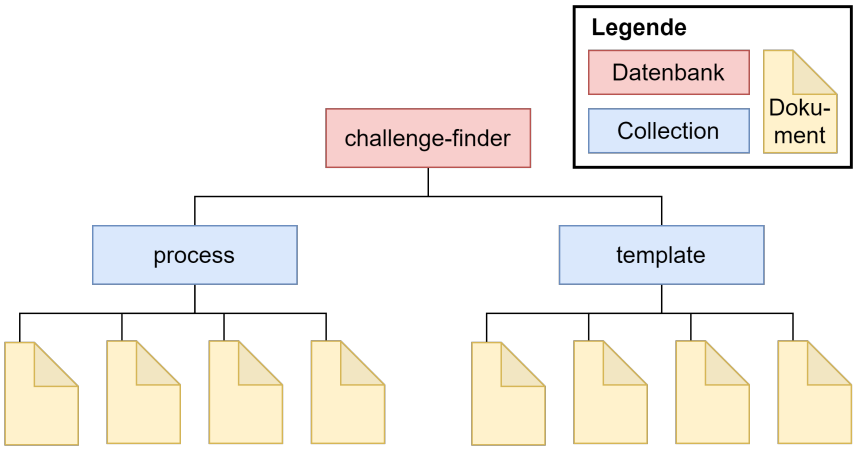

Abbildung 7.1: Struktur in MongoDB

Das Datenmodell für den Prozess zur Definition einer Design Challenge ergibt sich aus Abbildung 5.3 und Abbildung 7.2.

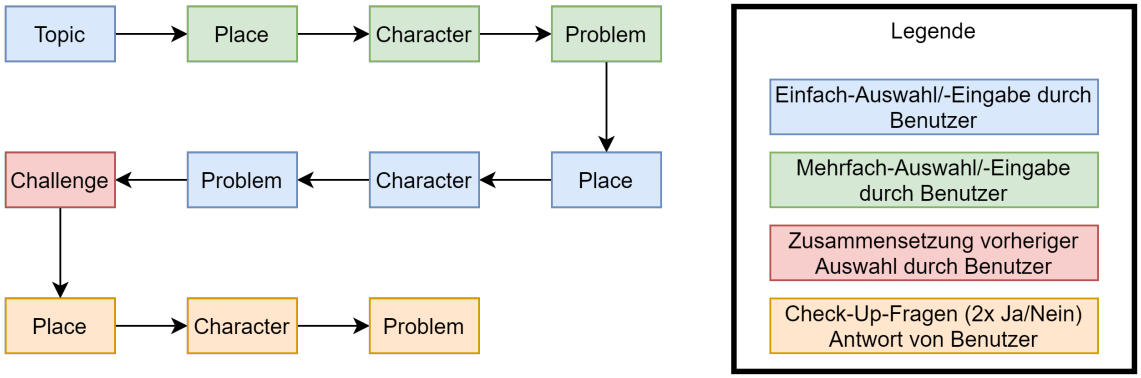

Abbildung 7.2: Ablauf des Prozesses mit Benutzereingabe

In Abbildung 5.3 und Abbildung 7.2 ist erkennbar, dass sich der Prozess in die folgenden Phasen bzw. Benutzereingaben gliedert:

- Thema
- Ideensammlung/Brainstorming
- konkrete Auswahl (Fokus)
- Formulierung der Challenge
- Checkup-Fragen

„MongoDB ist eine Dokumentdatenbank, die Daten in einem JSON-ähnlichen Dokumentformat speichert.“[31] Somit wird das Datenmodell in JavaScript Object Notation (JSON) modelliert. Die oben genannten Phasen lassen sich somit direkt in die folgenden JSON-Felder überführen:

- topic
- brainstorm
- selection
- challenge
- checkup

Des Weiteren verfügt das Datenmodell über die Felder „_id“ und „step“. Das Feld „_id“ dient dabei zur eindeutigen Zuordnung eines Dokuments zu einem Prozess. Im Feld „step“ ist der Fortschritt des Prozesses hinterlegt, um diesen unterbrechen zu können und später zu diesem Stand zurückzukehren.

Die Felder „topic“ und „challenge“ haben den Datentyp string. Bei den Feldern „brainstorm“, „selection“ und „checkup“ hingegen handelt es sich um eingebettete Beziehungen bzw. Objekte. Hier ist zwischen den Kategorien „place“, „character“ und „problem“ zu unterscheiden, sodass diese jeweils Felder der Objekte sind.

Wie in Abbildung 7.2 dargestellt, handelt es sich bei den Kategorien des „brainstorm“-Objektes um eine Mehrfachauswahl bzw. -eingabe durch den Benutzer. Dadurch sind diese Felder Arrays vom Datentyp string. Bei den Kategorien des „selection“-Objektes hingegen handelt es sich um eine Einfachauswahl bzw. -eingabe, sodass hier jeweils der Datentyp string zum Einsatz kommt. Aufgrund dessen, dass in der Checkup-Phase pro Kategorie zwei Fragen vom Benutzer zu beantworten sind, ist jede Kategorie wiederum als eingebettetes Objekt dargestellt. Jedes dieser Objekte hat die Felder „multiple“, für die Antwort der ersten Frage, und „similarities“, für die Antwort der zweiten Frage. Da es sich um Ja/Nein-Fragen handelt sind diese Felder vom Datentyp bool. Beispielhaft ist dieses Datenmodell auch als JSON-Datei im „docs“-Verzeichnis in [29] enthalten.

Die Implementierung dieses Datenmodells, wie in Abbildung 7.3 dargestellt, erfolgt in Go mit Hilfe von Strukturen, auch structs genannt. Das gesamte Datenmodell für den Prozess ist dabei in der Struktur „CfProcess“ abgebildet. Diese hat mehrere Strukturen eingebettet, um die eingebetteten Beziehungen darzustellen. Die in der Struktur verwendeten Datentypen sind mit den Datentypen aus dem JSON-Modell identisch.

Derzeit verfügt das Datenmodell für den Prozess über keine Beziehung zur verwendeten Vorlage. Somit sollte bei der Weiterentwicklung ein Feld für eine verweisbasierte Beziehung hinzugefügt werden, dass z.B. die „_id“ der genutzten Vorlage enthält. Weiterhin ist das Speichern von Änderungsdatum und -uhrzeit denkbar, um einen historischen Verlauf zu erstellen.

Das Datenmodell für die Vorschläge bzw. Vorlagen lässt sich mit Hilfe des Anwendungsfalls aus Abschnitt 5.2 und dem Prozessablauf in Abbildung 5.3 ermitteln. Dabei ist erkennbar, dass Vorschläge nur bei der Themenauswahl und Ideensammlung anzuzeigen sind. Die Ideensammlung bzw. Brainstorming-Phase gliedern sich in die Kategorien Handlungsort, Charaktere und potentielle Probleme.

Daraus ergeben sich die folgenden Felder für das JSON-Modell:

- topic
- place
- character
- problem

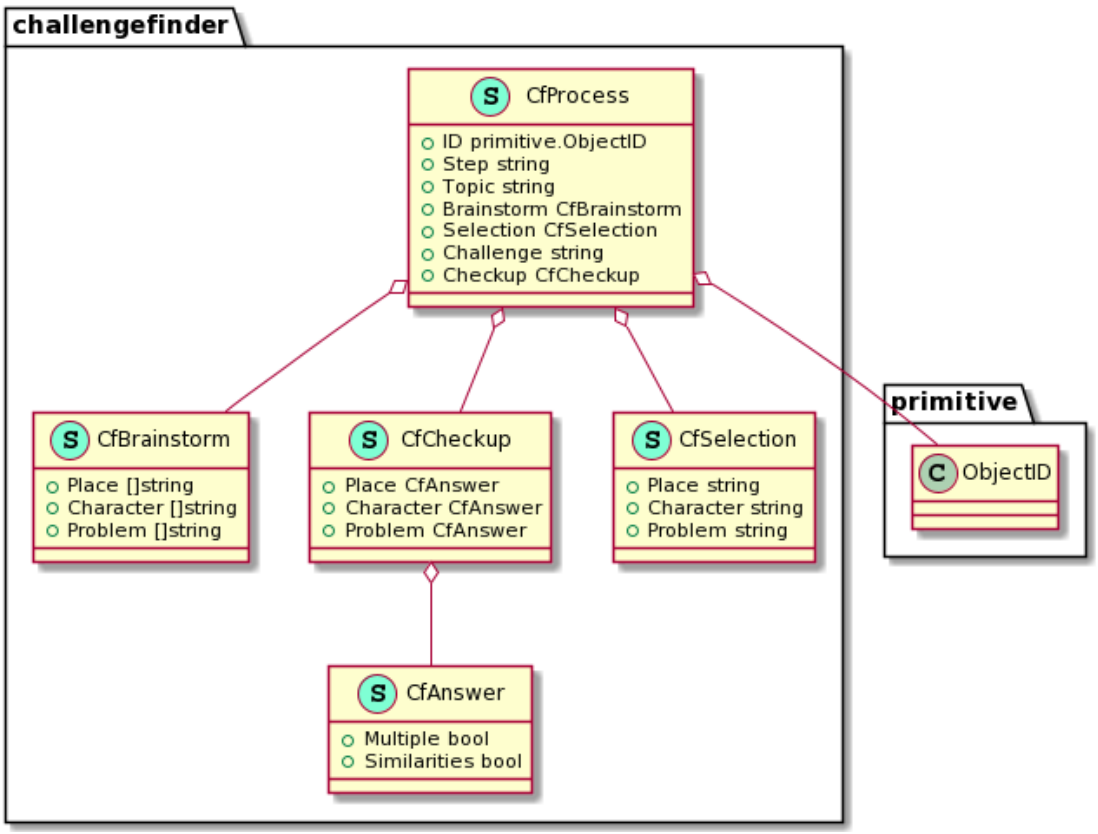

Abbildung 7.3: UML-Diagramm: Struktur „CfProcess“

Des Weiteren verfügt das Datenmodell über das Feld „_id“ zur eindeutigen Zuordnung des Dokuments. Alle Felder, bis auf „_id“, sind Arrays vom Datentyp string. Auch dieses Datenmodell ist als JSON-Beispiel im „docs“-Verzeichnis in [29] enthalten.

Die Implementierung in Go, wie auch in Abbildung 7.4 grafisch dargestellt, ist mit Hilfe einer Struktur umgesetzt. Diese hat den Namen „CfTemplate“. Die Namen der Felder der Struktur „CfTemplate“ sind identisch mit den Namen im JSON-Modell.

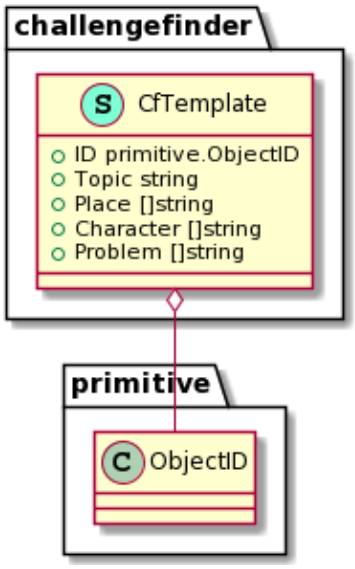

Abbildung 7.4: UML-Diagramm: Struktur „CfTemplate“

7.3 Implementierung von CRUD-Operationen

Für die Implementierung von CRUD-Operationen ist zunächst der Verbindungsaufbau zur MongoDB Datenbank notwendig. Dies ist mit Hilfe von Paketen des offiziellen „mongo-go-driver“[32] umgesetzt. Die Pakete bieten Möglichkeiten für CRUD-Operationen und den Verbindungsaufbau. Die Verbindung wird beim Start der Applikation aufgebaut und vor dem Beenden wieder geschlossen. Beim Verbindungsaufbau wird ein Zeiger erzeugt, der auf die Datenbank-Verbindung zeigt. Mit diesem lassen sich wiederum Zeiger erzeugen, die auf die gewünschte Collection zeigen. Collection-Zeiger sind notwendig, um mit Hilfe der Pakete aus [32] CRUD-Operationen durchzuführen. Die Anwendung der Pakete aus [32] ist grundlegend in den „Quick Start“-Anleitungen, in [13], [14], [15] und [16], beschrieben. Alle in den folgenden Absätzen beschriebenen Funktionen sind im Paket „challengefinder“ implementiert und befinden sich im gleichnamigen Verzeichnis in [29].

Aus dem Usecase in Abschnitt 5.2 geht hervor, dass die folgenden Operationen benötigt werden:

- Erstellen eines neuen Prozesses
- Laden eines vorhandenen Prozesses
- Thema festlegen
- Idee beim Brainstorming hinzufügen oder entfernen
- Fokus bzw. Selektion festlegen
- Challenge festlegen
- Checkup-Antworten festlegen
- nächster Schritt im Prozess

Grundsätzlich ist die Idee bei der Implementierung den Prozess sowohl temporär als Struktur, als auch dauerhaft als Dokument in der Datenbank zu speichern. Somit ist Programm-Logik mit der Struktur ausführbar, ohne dabei immer aus der Datenbank zu lesen. Kommt es zu Änderungen, so werden diese auf das Dokument in der Datenbank übertragen. Zu Beginn der Entwicklung enthielt die „CfProcess"-Struktur noch einen Collection-Zeiger. Jedoch stellte sich heraus, dass dieser als leeres Feld in der Datenbank angezeigt wird. Da dies nicht vorgesehen ist, dient die Struktur „CfDocument", wie in Abbildung 7.5 dargestellt, als Container. Diese enthält einen Collection-Zeiger und die Struktur „CfProcess", sodass nur der Prozess in der Datenbank gespeichert wird und keine ungenutzten leeren Felder entstehen.

Die zuvor aufgelisteten Operationen bzw. Funktionen lassen sich in folgende fünf Kategorien abstrahieren:

1. neuen Prozess und dazugehöriges Dokument erstellen
2. Laden eines Prozesses aus einem vorhandenen Dokument
3. Prozess: Variable vom Typ string setzen
4. Prozess: Array bzw. Slice vom Typ string setzen
5. Prozess: Variable vom Typ bool setzen

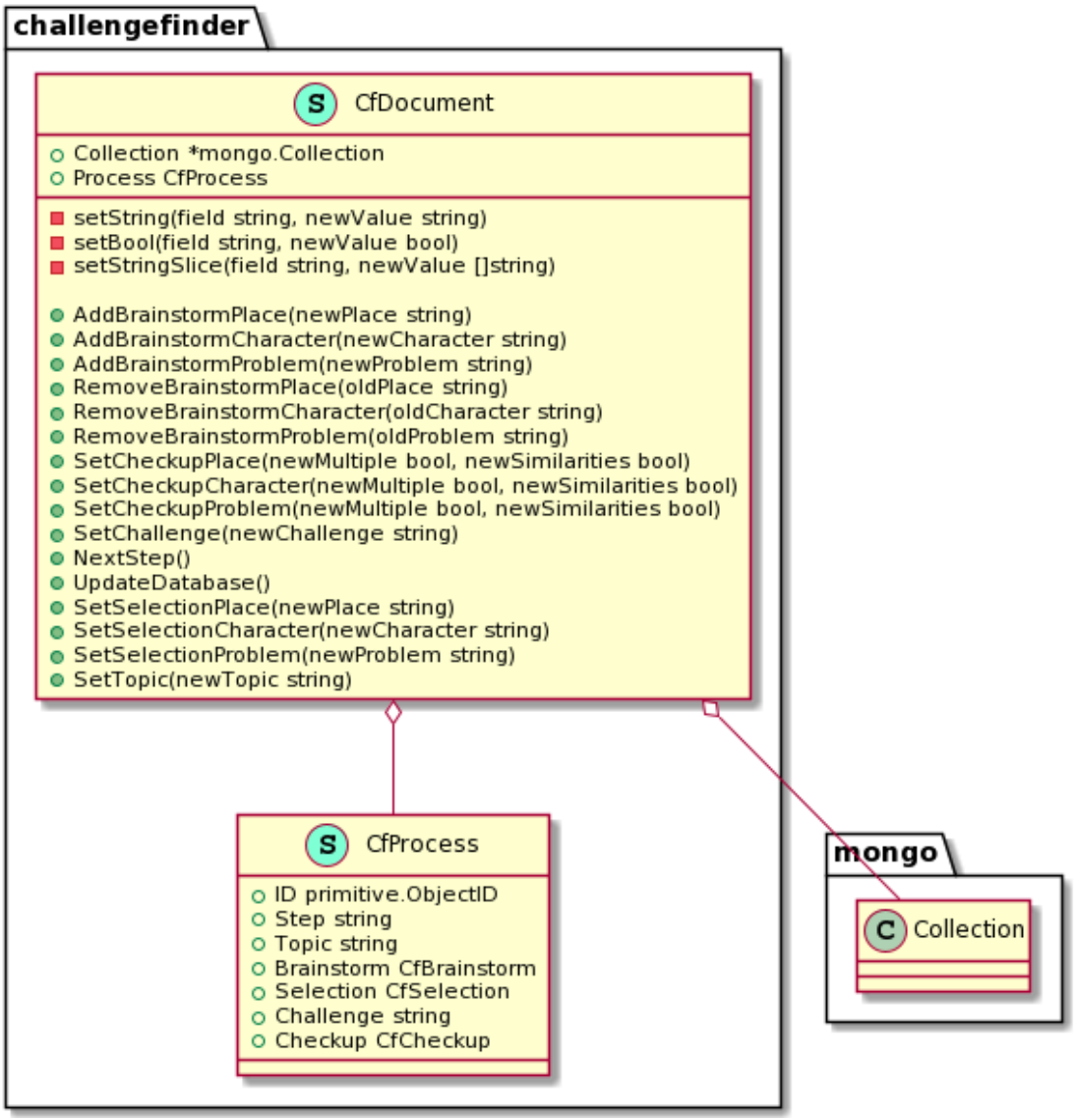

Abbildung 7.5: UML-Diagramm: Struktur „CfDocument“

Dabei ist erkennbar, dass die Funktionen der Kategorien 1 und 2 eine Struktur vom Typ „CfDocument“ als Rückgabewert haben. Diese Funktionen sind aufgrund der Großschreibung des Namen exportiert und dadurch auch außerhalb des Paketes aufrufbar. Bei Kategorie 1 wird zunächst eine solche Struktur erzeugt, um den Prozess abzubilden. Anschließend wird ein Dokument mit den Initialparametern des Prozesses in der Datenbank erzeugt. Kategorie 2 hingegen erwartete zusätzlich die „ID“ des bereits vorhandenen Dokuments als Parameter. Somit wird das Dokument aus der Datenbank gelesen und der Prozess in eine Struktur vom Typ „CfDocument“ übertragen. Sowohl bei Kategorie 1, als auch bei Kategorie 2, dient die Struktur als Rückgabewert, sodass diese für die weitere Programm-Logik zur Verfügung steht.

Kategorien 3, 4 und 5 sind hingegen Methoden mit Zeigerempfänger, wobei der Zeiger auf eine Struktur vom Typ „CfDocument“ zeigt. Diese Funktionen dienen zur Abstraktion und sind aufgrund der Kleinschreibung des Namen nicht exportiert. Somit ist ein Aufruf nur innerhalb des Paketes möglich. Dies ist auch im UML-Diagramm, in Abbildung 7.5, durch die Kennzeichnung mit einem roten Quadrat erkennbar. Weiterhin erwarten diese Funktionen den Namen des zu bearbeitenden Feldes, sowie den neuen Wert als Parameter. Hierbei wird der Wert des übergebenen Feldes in der Datenbank mit dem neuen Wert aktualisiert. Die Funktionen dienen zur Abstraktion und werden ausschließlich von Funktionen innerhalb des Paketes genutzt.

Weiterhin sind für jedes Feld einzelne „Setter“-Funktionen implementiert, um eine intuitive Nutzung zu ermöglichen. Die Funktionen sind, wie auch in Abbildung 7.5 durch einen grünen Kreis erkennbar, exportiert und somit auch außerhalb des Paketes nutzbar. Auch die „Setter“-Funktionen sind Methoden mit Zeigerempfänger vom Typ „CfDocument“ und erwarten als zusätzliche Parameter lediglich den zu setzenden bzw. zu entfernenden Wert. Dadurch ist es möglich, Dokumente in der Datenbank zu bearbeiten, ohne jegliches Wissen zur Datenbank selbst. Die „Setter“-Funktionen rufen im Allgemeinen eine der abstrahierten Funktionen, je nach Datentyp, zum Editieren von Dokumenten der Datenbank auf. Anschließend wird die Änderung auf die Struktur übertragen.

Die soeben beschriebenen und in Abbildung 7.5 dargestellten Funktionen bieten derzeit Grundfunktionalitäten. Somit ist die Umsetzung eines Prototypen möglich. Jedoch sollten im späteren Verlauf Funktionen mit folgenden Funktionalitäten umgesetzt werden:

- vorheriger Schritt
- Schritt setzen (Möglichkeit zur Iteration des Prozesses)
- neue Vorlage speichern
- Vorschläge zu vorhandener Vorlage hinzufügen
- automatische Formulierung einer Design Challenge auf Basis der Auswahl/„selection“

7.4 Testen der CRUD-Operationen

Dieser Abschnitt umfasst das Testen der implementierten CRUD-Operationen. Alle Tests befinden sich im Verzeichnis „test“ in [29]. Tests dienen zur Prüfung der Funktionalität von Funktionen und helfen dabei Fehler aufzudecken. Go verfügt über eine eingebaute Unterstützung von Unit-Tests, welche das Testen während der Entwicklung oder im laufenden Betrieb erleichtern. Mit Hilfe von Namenskonventionen und dem Paket „testing“[7], sowie dem Befehl „go test“ lassen sich Unit-Tests schnell implementieren und ausführen. [7], [6]

Die eingebaute Unterstützung von Unit-Tests unterliegt einer Namenskonvention. Dateien, die Testfunktionen enthalten, müssen mit „_test.go“ enden. Diese Konvention teilt dem Befehl „go test“ mit, dass diese Dateien Testfunktionen enthalten. Des Weiteren sind die Testfunktionen nach der Konvention „TestName“ zu benennen, wobei „Name“ etwas über den jeweiligen Test aussagt (z.B. der Name der zu testenden Funktion). Zudem müssen diese Funktionen einen Zeiger vom Typ T, aus dem „testing“-Paket, als Parameter erwarten. Dieser Zeiger wird für die Rückmeldung und Protokollierung von Tests genutzt. Der Befehl „go test“ führt die Testfunktionen aus und gibt eine Rückmeldung, ob der Test bestanden wurde („PASS“) oder nicht („FAIL“). [7], [6]

Im Go-Projekt [29] sind für die exportierten Funktionen des „challengefinder“-Paketes Tests implementiert. Diese befinden sich im „test“-Verzeichnis [29]. Die Tests unterliegen der Einschränkung, dass diese nicht das Lesen, Schreiben und Erstellen von Dokumenten in der Datenbank testen. Diese testen lediglich die Funktionalitäten bzgl. der Struktur „CfDocument“. Bei der Weiterentwicklung des Backends sollten auch Tests für die Interaktion mit der Datenbank implementiert werden, um die korrekte Funktionsweise des Paketes sicherzustellen.

Neben Testfunktionen sind für Unit-Tests auch Test-Sets notwendig. Diese definieren die Eingabedaten und erwarteten Ausgabedaten. Beim Testen des „challengefinder“-Paketes handelt es sich bei den Daten um Strukturen vom Typ „CfDocument“. In den Testfunktionen werden die definierten Eingabedaten an die zu testende Funktion übergeben und die Ausgaben werden mit den erwarteten Ausgabedaten verglichen. Um den Vergleich der Struktur zu erleichtern, ist hierfür eine zusätzliche Funktion im Paket „challengefinder“ implementiert. Derzeit umfasst das Test-Set vier Strukturen, wodurch nicht alle möglichen Eingabewerte getestet werden. Um

die Funktionalitäten des „challengefinder“-Paketes bestmöglich auf Fehler zu prüfen, ist eine Erweiterung des Test-Sets im Verlauf der Weiterentwicklung sinnvoll.

Um die Prüfung der Funktionalität zu automatisieren, wurden die Unit-Tests als GitHub Action im Remote-Repository integriert. Die GitHub Action führt die Tests bei jedem Upload durch. Dabei wird nach dem Ende des Tests ein kurzes Ergebnis, wie in Abbildung 7.6 zu sehen, angezeigt.

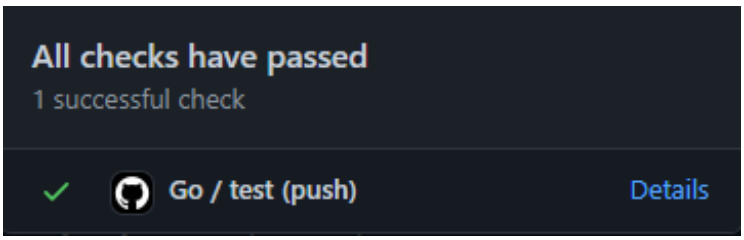

Abbildung 7.6: GitHub Action - Unit-Test

8 API

Die Anforderungen in Kapitel 2 umfassen eine API und die Möglichkeit zur kollaborativen Zusammenarbeit. Die Umsetzung ist im folgendem Kapitel thematisiert. Hierfür wird ein geeignetes Protokoll zur Datenübertragung evaluiert und implementiert. Zudem erfolgt die Definition der Schnittstelle und die Implementierung eines Anfragen-Handlers.

8.1 Evaluation eines geeigneten Protokolls

Zur Umsetzung der API und der Anforderung kollaborativer Zusammenarbeit ist zunächst ein geeignetes Protokoll für die Datenübertragung zwischen Client und Server zu wählen. Für die kollaborative Zusammenarbeit bieten sich bidirektionale Protokolle an, bei denen der Client keine Anfrage senden muss, um neue Nachrichten zu empfangen. Im Rahmen der Studienarbeit wurden hierbei gRPC und das WebSocket-Protokoll betrachtet.

Mit gRPC ist es einem Client möglich, direkt Funktionen, wie in Abbildung 8.1 dargestellt, auf einem anderen Rechner aufzurufen. Dabei erfolgt der Aufruf ähnlich wie bei einer lokalen Funktion. Die grundsätzliche Idee von gRPC ist die Definition eines Dienstes, der die aufrufbaren Server-Funktionen inklusive der Parameter und Rückgabewerte angibt. Die Schnittstelle ist im Regelfall mit Protocol Buffers definiert. Bei gRPC gibt es verschiedene Möglichkeiten, was beim Aufruf einer Server-Funktion geschieht. Für die API und kollaborative Zusammenarbeit sind jedoch nur bidirektionale streaming Remote Procedure Calls (RPC) relevant. Dabei können beide Seiten Nachrichten über einen Lese-Schreib-Stream senden. Beide Streams sind unabhängig voneinander und die Reihenfolge von Lesen und Schreiben ist anwendungsspezifisch. [33], [34]

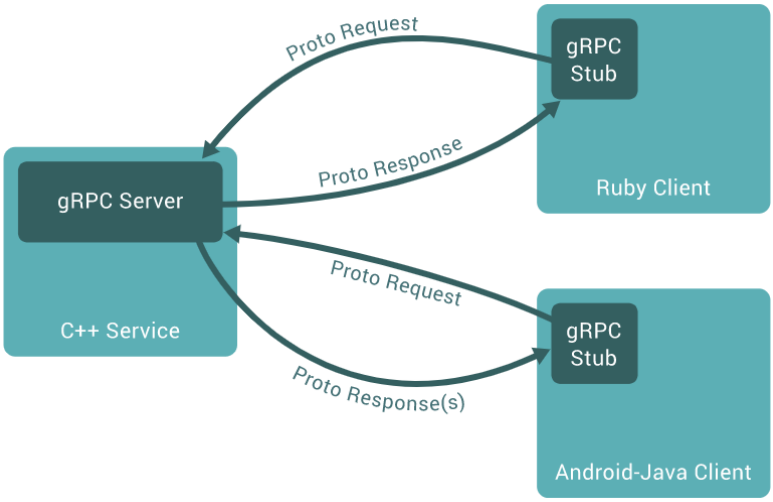

Abbildung 8.1: gRPC-Verbindung [33]

WebSocket ist ein Standard-Netzwerkprotokoll, dass den Austausch von Daten definiert. Der Standard ist in RFC 6455 [20] der IETF zu finden. Ziel des WebSocket-Protokolls ist die Bereitstellung eines Mechanismus zur bidirektionalen Kommunikation bei browserbasierten Anwendungen, ohne mehrere Hypertext Transfer Protocol (HTTP)-Verbindungen zu öffnen. Das Protokoll besteht aus einem eröffnenden Handshake, gefolgt von Nachrichten-Rahmen, und setzt auf dem Transmission Control Protocol (TCP) auf. Eine solche WebSocket-Verbindung ist in Abbildung 8.2 dargestellt. Bei dem Handshake werden alle nötigen Informationen zur Identifikation ausgetauscht und ein Kommunikationskanal eröffnet. Dieser Kommunikationskanal bleibt geöffnet, sodass der Server jederzeit Nachrichten, auch ohne Anfrage, an den Client senden kann. Somit kann der Server z.B. Nachrichten eines Clients direkt an alle verbundenen Clients weiterleiten, ohne dass diese eine Anfrage stellen. Das WebSocket-Protokoll kommt häufig bei Chat-Anwendungen zum Einsatz. Für die Implementierung in Go stehen verschiedene Pakete, wie z.B. „socketio“ [35] oder „websocket“ [21] zur Verfügung. Diese enthalten bereits Beispiele für die Umsetzung einer Chat-Anwendung. [19], [20]

Sowohl mit gRPC, als auch mit dem WebSocket-Protokoll ist die Umsetzung kollaborativer Zusammenarbeit und einer API möglich. Beide Protokolle stellen eine bidirektionale Kommunikation bereit. Hierbei sind bei gRPC zuvor konkrete Server-Funktionen zu definieren. Hingegen beim WebSocket-Protokoll sind beliebige Nachrichten versendbar. Somit ist das WebSocket-Protokoll flexibler, jedoch muss hier serverseitig ein Anfragen-Handler implementiert werden. Des Weiteren bieten die Beispiele einer Chat-Anwendung, in den Go Paketen zur Implementierung des

WebSocket-Protokolls, eine sehr gute Grundlage für die Umsetzung kollaborativer Zusammenarbeit und der API. Dabei ist die gesamte Kommunikation zwischen Server und Clients bereits implementiert und es ist lediglich eine API zu definieren und ein Anfragen-Handler für diese zu implementieren. Aufgrund dessen kommt das WebSocket-Protokoll zur Umsetzung der API und der Anforderung kollaborativer Zusammenarbeit zum Einsatz.

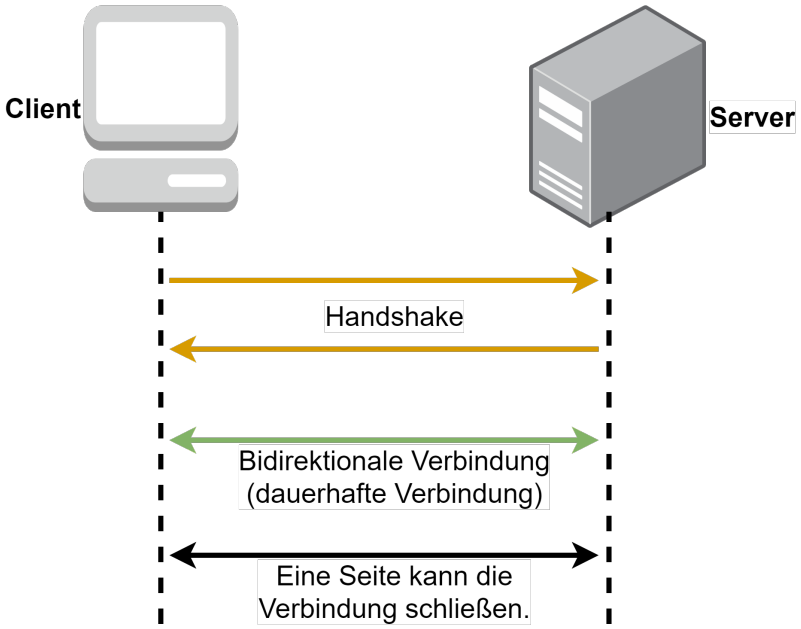

Abbildung 8.2: WebSocket-Verbindung

8.2 Implementierung des WebSocket-Protokolls

Wie bereits bei der Evaluation des Protokolls, in Abschnitt 8.1 erwähnt, ist in den Go Paketen „socketio“ [36] und „websocket“ [37] das Protokoll bereits implementiert. Des Weiteren verfügen beide über eine Chat-Anwendung als Beispiel. Die Idee bei der Umsetzung ist es, das Beispiel als Grundlage zu nutzen und mit einer API-Definition, in Form eines Anfragen-Handlers, zu ergänzen.

Zunächst wurde hierfür das Paket „socketio“ mit dem dazugehörigen Chat-Beispiel genutzt, da dieses über stark abstrahierte Funktionen verfügt. Jedoch stellte sich heraus, dass das Beispiel in der verwendeten Version (1.6.0) nur einen einzelnen Client unterstützt. Somit wurde die Verwendung des Paketes „socketio“ verworfen.

Da das Paket „websocket“ auch ein Chat-Beispiel enthält, wird dieses zur Implementierung genutzt. Die gesamte Umsetzung der API ist im Paket „collaboration“ [29], wie zuvor in Abschnitt 6.2 beschrieben, realisiert. Dabei ist der Code der Dateien „client.go“ und „hub.go“ dem Chat-Beispiel aus [37] entnommen und modifiziert. Im „collaboration“-Paket sind, wie in Abbildung 8.3 dargestellt, die beiden Strukturen „Client“ und „Hub“ definiert. [21], [37]

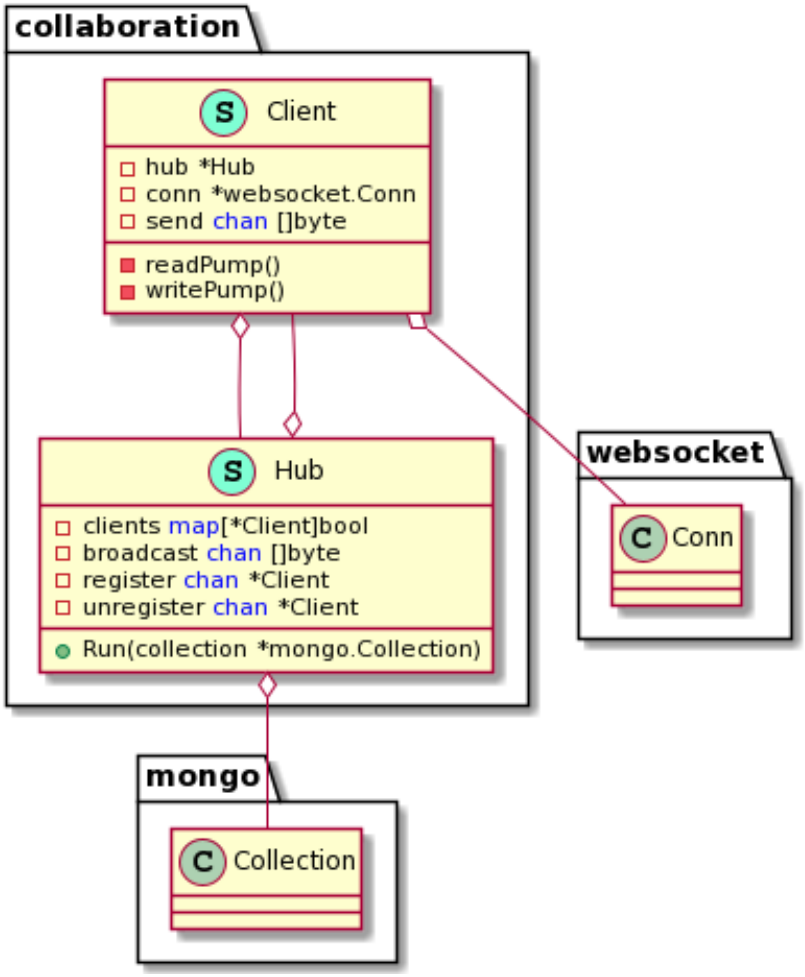

Abbildung 8.3: UML-Diagramm Paket „collaboration“

Für jede WebSocket-Verbindung wird eine „Client“-Struktur erzeugt. Diese ist der Vermittler zwischen einer WebSocket-Verbindung und einer einzelnen „Hub“-Struktur. Eine „Hub“-Struktur dient zur Verwaltung der Clients und sendet Nachrichten an diese. In der Anwendung wird für den Hub eine Go-Routine und für jeden Client werden zwei Go-Routinen ausgeführt. Dabei kommunizieren die Go-Routinen über Kanäle. Die Go-Routine des Hubs führt dabei die „Run“-Funktion aus. Bei einem Client dient eine Go-Routine zum Lesen und eine zum Schreiben von Nachrichten aus bzw. in den WebSocket. [21], [37], [29]

Die Implementierung des WebSocket-Protokolls umfasst keine unterschiedlichen Räume oder Sitzungen. Somit ist es derzeit nicht möglich, mehrere Design Challenges parallel zueinander zu definieren. Diese Funktion sollte bei der Weiterentwicklung des Backends mit in die Anforderungen aufgenommen werden, damit mehrere Teams gleichzeitig an unterschiedlichen Design Challenges arbeiten können. Eine mögliche Umsetzung ist bereits in einem Issue [38] des „websocket“-Paketes beschrieben.

8.3 Definition der API und Implementierung des Anfragen-Handlers

Dieser Abschnitt umfasst die Definition der API, sowie die Implementierung des Anfragen-Handlers. Grundlegend soll es Clients möglich sein, über die API, die zuvor in Abschnitt 7.3 implementierten Funktionen aufzurufen. Das Ergebnis soll anschließend an alle verbunden Clients gesendet werden.

Grundsätzlich sind die Anfragen und Antworten als JSON formatiert. Dieses Format ist sowohl für Menschen lesbar, als auch gut im Frontend verarbeitbar. Eine API-Anfrage hat die Felder „function“ und „parameter“.

Jede Anfrage muss das Feld „function“ enthalten. Hingegen dürfen nur Anfragen zu Funktionen, die auch Parameter erwarten, das Feld „parameter“ enthalten. Der Wert des „function“-Feldes wird als Typ string erwartet. Bei dem Feld „parameter“ kann der Wert vom Typ string oder ein Array vom Typ bool sein. Dies hängt von der jeweiligen angefragten Funktion ab. Stimmt der Typ des Wertes nicht mit dem erwarteten Typ überein, oder ist die angeforderte Funktion nicht aufrufbar, so wird eine Fehlermeldung zurückgeschickt. Für die API sind die zuvor in Abschnitt 7.3 beschriebenen CRUD-Operationen verfügbar. Wenn eine korrekte API-Anfrage, wie in Abbildung 8.4 zu sehen, den Server erreicht, wird die angefragte Funktion mit den übergebenen Parametern ausgeführt und das Ergebnis, als JSON, an alle Clients gesendet. [29]

Die Abwicklung der API-Anfragen ist in Form eines Handlers realisiert. Dieser nimmt die Anfrage entgegen und ordnet („Unmarshalling“) die Daten der JSON in einer Map. Anschließend werden die empfangenen Daten auf Korrektheit geprüft und entsprechend ein Fehler zurückgesendet oder die angeforderte Funktion ausge-

führt. Nach dem Ausführen einer Funktion wird das Ergebnis, des Design Challenge Prozesses, als JSON formatiert („Marshalling“) und zurückgegeben. [29], [39]

```
{
    "function": "SetCheckupPlace",
    "parameter": [true, true]
}
```

Abbildung 8.4: Beispiel einer API-Anfrage

Zum jetzigen Entwicklungsstand wird die API-Anfrage einer Map zugeordnet. Maps sind in der Handhabung leider sehr umständlich, sodass eine Struktur, bestehend aus einem Feld vom Typ string für den Funktionsnamen und einem Feld vom Typ Interface für die Parameter, geeigneter für die Verarbeitung von Anfragen ist. Die Umsetzung dieser Änderung ist bei der Weiterentwicklung wünschenswert.

8.4 Testen der API mit einem simplen Client

Zum Testen der API kommt der Client des Chat-Beispiels aus [37] zum Einsatz. Dabei handelt es sich um ein einfaches Frontend, bei dem der Benutzer die API-Anfrage in eine Textbox, wie in Abbildung 8.5 dargestellt, eingeben und an den Server senden kann. Die API-Antwort wird als Text, wie in Abbildung 8.6 zu sehen, in der Webanwendung angezeigt. Somit ist das händische Testen der API möglich.

Beim händischen Testen stellte sich heraus, dass die Fehlermeldungen zu falschen API-Anfragen an alle Clients gesendet werden. Hier wäre es sinnvoll, wenn nur der anfragende Client die Fehlermeldung erhält. Des Weiteren kam es zu einem Fehler bei der API-Anfrage der Funktion „LoadProcess“. Dabei war der Typ des Parameters richtig, jedoch existierte kein Dokument mit der übergebenen ID in der Datenbank. Dieser Fehler wird von der Funktion „LoadProcess“ erkannt, jedoch nicht an den Handler übergeben. Somit wird der Fehler auf dem Terminal ausgegeben und die Anwendung abgebrochen. Dieses Problem ist lösbar, indem Funktionen niedriger Abstraktionsebenen die auftretenden Fehler an die Funktionen höherer Abstraktionsebenen übergeben, anstatt diese lediglich auszugeben und das Programm abzubrechen.

Bei der Weiterentwicklung des Backends sollten die händischen Tests durch automatisierte Unit-Tests ersetzt werden, um systematisch zu testen und die bestmögliche Funktionalität der API zu gewährleisten.

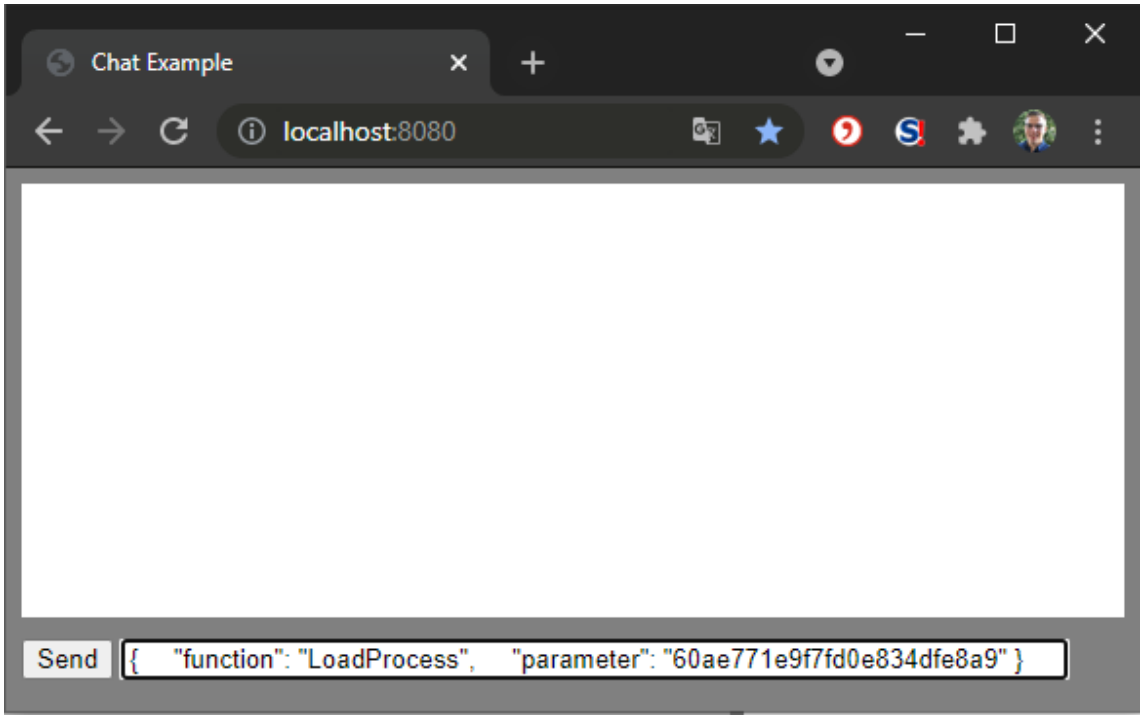

Abbildung 8.5: Client API-Anfrage

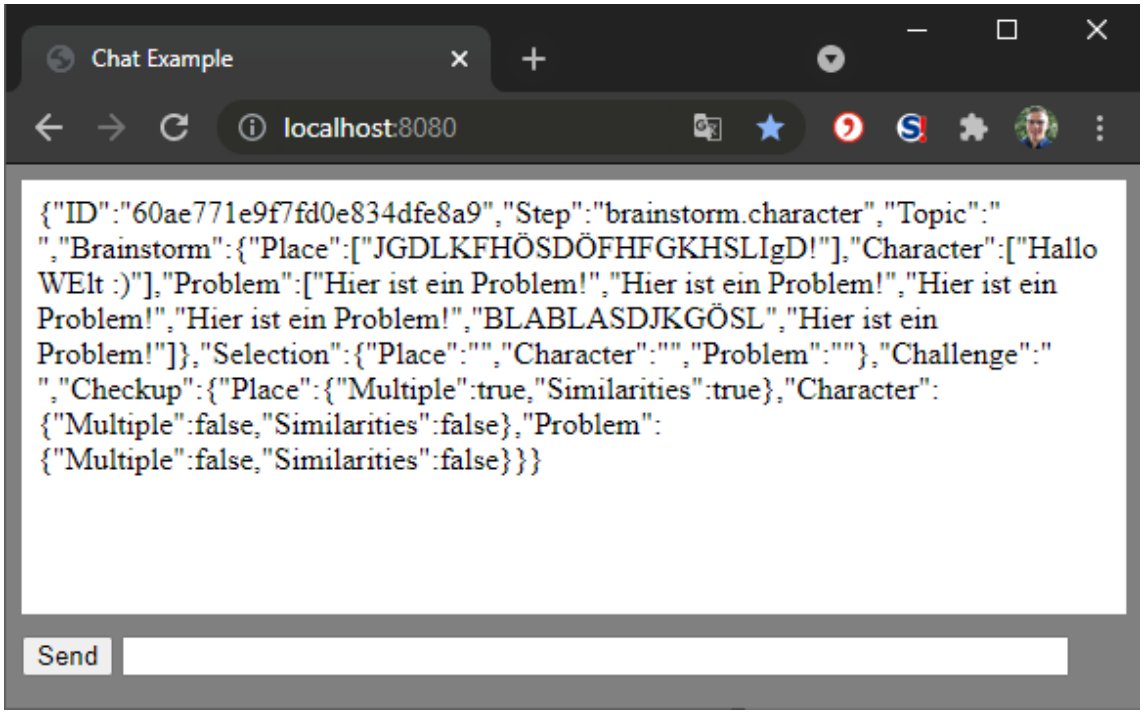

Abbildung 8.6: Client API-Antwort

9 Zusammenfassung

Dieses Kapitel der Studienarbeit, „Backend-Entwicklung eines kollaborativen Tools zur Definition von Design Challenges“, umfasst das Fazit und einen Ausblick. Dabei wird die Umsetzung des Backends reflektiert und es wird ein Ausblick gegeben, wie das Projekt zukünftig weitergeführt, erweitert und verbessert werden kann.

9.1 Fazit

Abschließend ist festzustellen, dass es im Zeitraum von 10 Wochen gelungen ist, die „Backend-Entwicklung eines kollaborativen Tools zur Definition von Design Challenges“ durchzuführen. Dabei wurde der Prozess zur Formulierung einer Design Challenge theoretisch betrachtet und ein Anwendungsfall für das Tool entwickelt. Zudem wurden die Programmiersprache für das Backend, eine Möglichkeit zur Datenspeicherung, sowie ein geeignetes Protokoll für die kollaborative Zusammenarbeit und die API evaluiert.

Das Backend ist mit der Programmiersprache Go umgesetzt, wobei das Projekt als Modul mit Paketen für die Datenspeicherung und die API aufgebaut ist. Zur Datenspeicherung kommt die dokumentenorientierte Datenbank MongoDB zum Einsatz. Für diese wurde jeweils ein Datenmodell für den Prozess und ein Datenmodell für Vorlagen entwickelt. Die Datenmodelle sind auch in Go mit Hilfe von Strukturen abgebildet. Um den Design Challenge Prozess auch in der Datenbank speichern zu können, sind CRUD-Operationen implementiert. Zum Testen dieser sind zudem Unit-Tests erstellt. Um die bestmögliche Funktionalität der umgesetzten Funktionen zu gewährleisten, sollten die Test-Sets erweitert werden. Zudem sind noch keine CRUD-Operationen zum Hinzufügen von Vorschlägen bzw. Vorlagen implementiert. Des Weiteren übergeben die Funktionen, zur Umsetzung der CRUD-Operationen,

Fehler nicht an höhere Abstraktionsebenen. Dies führt dazu, dass Fehler, wie z.B. das Laden eines nicht vorhandenen Dokuments aus der Datenbank, lediglich über das Terminal ausgegeben werden und die Anwendung abbricht.

Die Anforderung der kollaborativen Zusammenarbeit und der API ist mit Hilfe des WebSocket-Protokolls umgesetzt. Dieses ist mit dem „websocket“-Paket implementiert, sodass es Clients möglich ist, Anfragen an den Server zu senden und Antworten ohne explizite Anforderung zu erhalten. Des Weiteren ist eine API definiert und ein Anfragen-Handler implementiert. Mit Hilfe der API können Clients CRUD-Operationen beim Server anfragen. Das Ergebnis wird an alle verbundenen Clients gesendet. Bei der derzeitigen Implementierung des Protokolls sind alle Clients in derselben Sitzung, sodass alle Clients an derselben Design Challenge arbeiten. Hierbei wäre die Realisierung von Sitzungen bzw. Räumen sinnvoll. Somit könnten mehrere Teams parallel zueinander an verschiedenen Design Challenges arbeiten. Weiterhin ist die serverseitige Abbildung der API-Anfrage als Map umgesetzt, wodurch die Weiterarbeit komplexer ist. Dies kann durch die Verwendung einer Struktur vereinfacht werden. Das Testen der API wird derzeit händisch mit Hilfe eines einfachen Clients durchgeführt. Hier sollten zukünftig Unit-Tests implementiert werden, um Funktionen und die API systematischer zu testen.

Grundsätzlich wurden die Anforderungen aus Kapitel 2 erfüllt. Dabei besteht bei der Implementierung, wie zuvor beschrieben, noch Verbesserungs- und Erweiterungspotential.

9.2 Ausblick

Das Backend des Tools zur Definition von Design Challenges verfügt bereits über grundlegende Funktionalitäten, sodass dieses als Prototyp eingesetzt werden kann. Die Entwicklung des Tools wird nach Abgabe der Studienarbeit mit Maximilian Kürschner, mit dem die Idee während eines Hackathons entstand, weitergeführt. Dabei ist es sinnvoll, die zuvor thematisierten Fehler und Erweiterungsmöglichkeiten zu beheben und umzusetzen.

Kurzfristig ist die Umsetzung eines ersten funktionsfähigen Prototypen geplant. Hierfür ist die Entwicklung eines Frontends notwendig. Dabei sollen Hilfestellungen

zur Anwendung der Methoden und dem Prozess bereitgestellt werden, sodass eine intuitive Definition einer Design Challenge möglich ist. Mittelfristig ist die Implementierung verschiedener Methoden zur Formulierung geplant, sodass der kreative Prozess unterstützt wird. Langfristiges Ziel des Projektes ist die Entwicklung einer benutzerfreundlichen und selbsterklärenden Web-Anwendung, um den gesamten Design-Thinking-Prozess abzubilden. Dabei ist die grundlegende Anforderung, den theoretischen Prozess so weit zu abstrahieren, dass jeder Benutzer den Prozess und die Methoden, ohne Vorwissen, umsetzen bzw. anwenden kann.

Abbildungsverzeichnis

Literaturverzeichnis

[1] R. Simschek und F. Kaiser, *Design Thinking: Innovationen effektiv managen*. München: UVK Verlag, 2019, ISBN: 9783739880105. Adresse: `https://elibrary.narr.digital/book/99.125005/9783739880105`.

[2] R. Lahdo, *Startpunkt: Herausforderung festlegen*, Mannheim, 2020, Verfügbar in der digitalen Abgabe: Quellen/Lahdo-Ideation.pdf.

[3] M. Kürschner und J. Schackniß. „UnMaze: Damit die Challenge-Formulierung nicht zur Challenge wird!“ (2021), Adresse: `https://github.com/programonaut/mesh-2021` (besucht am 30.05.2021). Verfügbar in der digitalen Abgabe: Quellen/Kuerschner_Schackniss-Hackathon.zip.

[4] Google, Hrsg. „A Tour of Go“. (2021), Adresse: `https://tour.golang.org/` (besucht am 08.06.2021).

[5] Google, Hrsg. „The Go Programming Language: Documentation“. (2021), Adresse: `https://golang.org/doc/` (besucht am 03.06.2021).

[6] Google, Hrsg. „The Go Programming Language: Add a test“. (), Adresse: `https://golang.org/doc/tutorial/add-a-test` (besucht am 06.06.2021).

[7] Google, Hrsg. „The Go Programming Language: testing“. (), Adresse: `https://golang.org/pkg/testing/` (besucht am 06.06.2021).

[8] Git, Hrsg. „Git - Documentation“. (2021), Adresse: `https://git-scm.com/doc` (besucht am 08.06.2021).

[9] GitHub, Hrsg. „GitHub Documentation“. (2021), Adresse: `https://docs.github.com/en` (besucht am 08.06.2021).

[10] L. Piepmeyer, *Grundkurs Datenbanksysteme: Von den Konzepten bis zur Anwendungsentwicklung*. München und Wien: Hanser, 2011, ISBN: 3446423540.

[11] S. Edlich, *NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*, 2., aktualisierte und erw. Aufl. München: Hanser, 2011, ISBN: 3446427538.

[12] MongoDB, Hrsg. „NoSQL Databankenerklärung“. (2021), Adresse: `https://www.mongodb.com/de-de/nosql-explained` (besucht am 04.06.2021).

[13] N. Raboy. „Quick Start: Golang & MongoDB - Starting and Setup“. MongoDB, Hrsg. (2019), Adresse: https://www.mongodb.com/blog/post/quick-start-golang-mongodb-starting-and-setup (besucht am 05. 06. 2021).

[14] ——, „Quick Start: Golang & MongoDB - How to Create Documents“. MongoDB, Hrsg. (2019), Adresse: https://www.mongodb.com/blog/post/quick-start-golang--mongodb--how-to-create-documents (besucht am 05. 06. 2021).

[15] ——, „Quick Start: Golang & MongoDB - How to Update Documents“. MongoDB, Hrsg. (2019), Adresse: https://www.mongodb.com/blog/post/quick-start-golang--mongodb--how-to-update-documents (besucht am 05. 06. 2021).

[16] ——, „Quick Start: Golang & MongoDB - Modeling Documents with Go Data Structures“. MongoDB, Hrsg. (2020), Adresse: https://www.mongodb.com/blog/post/quick-start-golang--mongodb--modeling-documents-with-go-data-structures (besucht am 05. 06. 2021).

[17] MongoDB, Hrsg. „The MongoDB 4.4 Manual“. (2020), Adresse: https://docs.mongodb.com/manual/ (besucht am 08. 06. 2021).

[18] gRPC, Hrsg. „Documentation“. (2021), Adresse: https://grpc.io/docs/ (besucht am 08. 06. 2021).

[19] 1&1 IONOS SE, Hrsg. „Was ist WebSocket?“ (2020), Adresse: https://www.ionos.de/digitalguide/websites/web-entwicklung/was-ist-websocket/ (besucht am 08. 06. 2021).

[20] A. Melnikov und I. Fette, *The WebSocket Protocol*, IETF, Hrsg., RFC 6455, Dez. 2011. DOI: 10.17487/RFC6455. Adresse: https://rfc-editor.org/rfc/rfc6455.txt.

[21] Gorilla Web Toolkit, Hrsg. „websocket“. (2020), Adresse: https://pkg.go.dev/github.com/gorilla/websocket (besucht am 08. 06. 2021).

[22] GitHub, Hrsg. „GitHub features: Integrated project management tools“. (2021), Adresse: https://github.com/features/project-management/ (besucht am 31. 05. 2021).

[23] „JavaScript (Informatik)“, in *Brockhaus Enzyklopädie Online*, NE GmbH | Brockhaus, 2021. Adresse: https://brockhaus.de/ecs/permalink/3E1A3B455B26DD03553D34CCD0B49B7F.pdf.

[24] OpenJS Foundation, Hrsg. „Über Node.js | Node.js“. (2021), Adresse: https://nodejs.org/de/about/ (besucht am 03. 06. 2021).

[25] Intellectsoft US, Hrsg. „NodeJs vs Golang: Which is Best for Backend Development?“ (2018), Adresse: `https://www.intellectsoft.net/blog/nodejs-vs-golang/` (besucht am 03. 06. 2021).

[26] Google, Hrsg. „The Go Programming Language: The Go Project“. (2021), Adresse: `https://golang.org/project` (besucht am 03. 06. 2021).

[27] Google, Hrsg. „The Go Programming Language: Effective Go“. (2021), Adresse: `https://golang.org/doc/effective_go` (besucht am 03. 06. 2021).

[28] Google, Hrsg. „The Go Programming Language: Tutorial: Create a Go module“. (2021), Adresse: `https://golang.org/doc/tutorial/create-module` (besucht am 03. 06. 2021).

[29] J. Schackniß. „challengefinder-backend: Studienarbeit II - Backend-Entwicklung eines kollaborativen Tools zur Definition von Design Challenges“. (2021), Adresse: `https://github.com/schackniss/challengefinder-backend` (besucht am 02. 06. 2021). Tag: 0.1.0. Verfügbar in der digitalen Abgabe: challengefinder-backend-0.1.0.zip.

[30] R. Adams, *SQL: Der Grundkurs für Ausbildung und Praxis. Mit Beispielen in MySQL/MariaDB, PostgreSQL und T-SQL*, 3., aktualisierte Auflage. München: Carl Hanser Verlag GmbH & Co. KG und Ciando, 2019, ISBN: 9783446462748.

[31] MongoDB, Hrsg. „Die beliebteste Datenbank für moderne Apps“. (2021), Adresse: `https://www.mongodb.com/de-de` (besucht am 05. 06. 2021).

[32] MongoDB, Hrsg. „mongo-go-driver“. (2021), Adresse: `https://github.com/mongodb/mongo-go-driver` (besucht am 05. 06. 2021).

[33] gRPC. „Introduction to gRPC“. (2021), Adresse: `https://grpc.io/docs/what-is-grpc/introduction/` (besucht am 09. 06. 2021).

[34] ——, „Core concepts, architecture and lifecycle“. (2021), Adresse: `https://grpc.io/docs/what-is-grpc/core-concepts/` (besucht am 09. 06. 2021).

[35] G. Lee. „socketio“. (2021), Adresse: `https://pkg.go.dev/github.com/googollee/go-socket.io` (besucht am 09. 06. 2021).

[36] ——, „go-socket.io“. (2021), Adresse: `https://github.com/googollee/go-socket.io` (besucht am 10. 06. 2021). Verfügbar in der digitalen Abgabe: Quellen/Lee-socketio.zip.

[37] Gorilla Web Toolkit. „websocket“. (2021), Adresse: `https://github.com/gorilla/websocket` (besucht am 10. 06. 2021). Verfügbar in der digitalen Abgabe: Quellen/Gorilla-websocket.zip.

[38] „Creating chat 'rooms' · Issue #46 · gorilla/websocket“. (2014), Adresse: `https://github.com/gorilla/websocket/issues/46` (besucht am 10. 06. 2021).

[39] A. Gerrand. „The Go Blog: JSON and Go“. Google, Hrsg. (2011), Adresse: `https://blog.golang.org/json` (besucht am 12. 06. 2021).