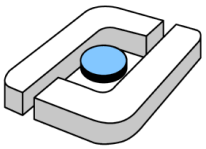


Hochschule Osnabrück

University of Applied Sciences

Fakultät
Ingenieurwissenschaften und Informatik

Schriftliche Ausarbeitung zum Thema:

Wohngemeinschafts-Verwaltung als verteilte Anwendung

im Rahmen des Moduls
Software-Architektur – Konzepte und Anwendungen,
des Studiengangs Informatik-Medieninformatik

Autor:	Tim Cirksena Maximilian Jaesch
Matr.-Nr.:	950976, 966282
E-Mail:	tim.cirksena@hs-osnabrueck.de maximilian.jaesch@hs-osnab-rueck.de
Themensteller:	Prof. Dr. Rainer Roosmann

Abgabedatum: 17.02.2023

Inhalt

Abbildungsverzeichnis

Die Abbildungen eines Anhangs werden nicht im Abbildungsverzeichnis aufgeführt. Dieses gilt analog für das Tabellenverzeichnis.

Source-Code Verzeichnis

Abkürzungsverzeichnis

CDI	Context and Dependency Injection for the Java EE Platform
JSON	JavaScript Object Notation
JsonB	JavaScript Object Notation Binary
ECB	Entity-Controller-Boundary Pattern
EJB	Enterprise Java Beans
Java EE	Java Enterprise Edition, in der Version 7
JSF	Java Server Faces
SWA	Software-Architektur
SFLB	Stateful Session Bean
SLSB	Stateless-Session Bean
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
JS	Java Script

[Anm.] In das Abkürzungsverzeichnis werden alle Abkürzungen aufgenommen, die nicht allgemein gebräuchlich sind (oder nicht im Duden stehen). Abkürzungen wie „etc.“, „z. B.“ und „z. Zt.“ gehören nicht in das Verzeichnis.

1 Einleitung

Autor: Tim Cirksena

Im heutigen Leben eines Studenten gibt es viele Probleme, das Wohnen in einer WG sollte keins davon sein. Jedoch treten immer wieder organisatorische Probleme auf, die durch Planung einfach behoben werden könnten. Probleme könnten folgende sein:

- Unangekündigter Besuch in der Klausurenphase.
- Der/die Mitbewohner/in besetzt die Küche und es kann kein Essen gekocht werden.
- Der/die Mitbewohner/in wird im Supermarkt getroffen, und es wurde ohne Absprache derselbe Einkauf getätigt.

Deshalb ist die Folgerung des erkannten Problems eine WG-Verwaltung in dem es Funktionalitäten gibt, die eine Lösung bieten. Unteranderem könnte das ein Kalender sein, indem Mitbewohner Termine austauschen können. Oder eine Einkaufsliste in der der gemeinsame Einkäufe geplant werden können.

1.1 Vorstellung des Themas

Autor: Tim Cirksena

Das Thema der Projektarbeit befasst sich mit der WG-Verwaltung. Ein vorläufiger Entwurf wurde erstellt, wie eine WG-Verwaltung aussehen könnte. Der Prototyp soll einen Überblick verschaffen was zu erwarten ist. Die verschiedenen Bereiche werden im späteren Verlauf des Berichts näher und genauer erklärt. Die Verwaltung dient einen organisatorischen Zweck, und somit zu helfen verschiedenste Probleme die damit verbunden sind zu lösen.

1.1.1 Mockup

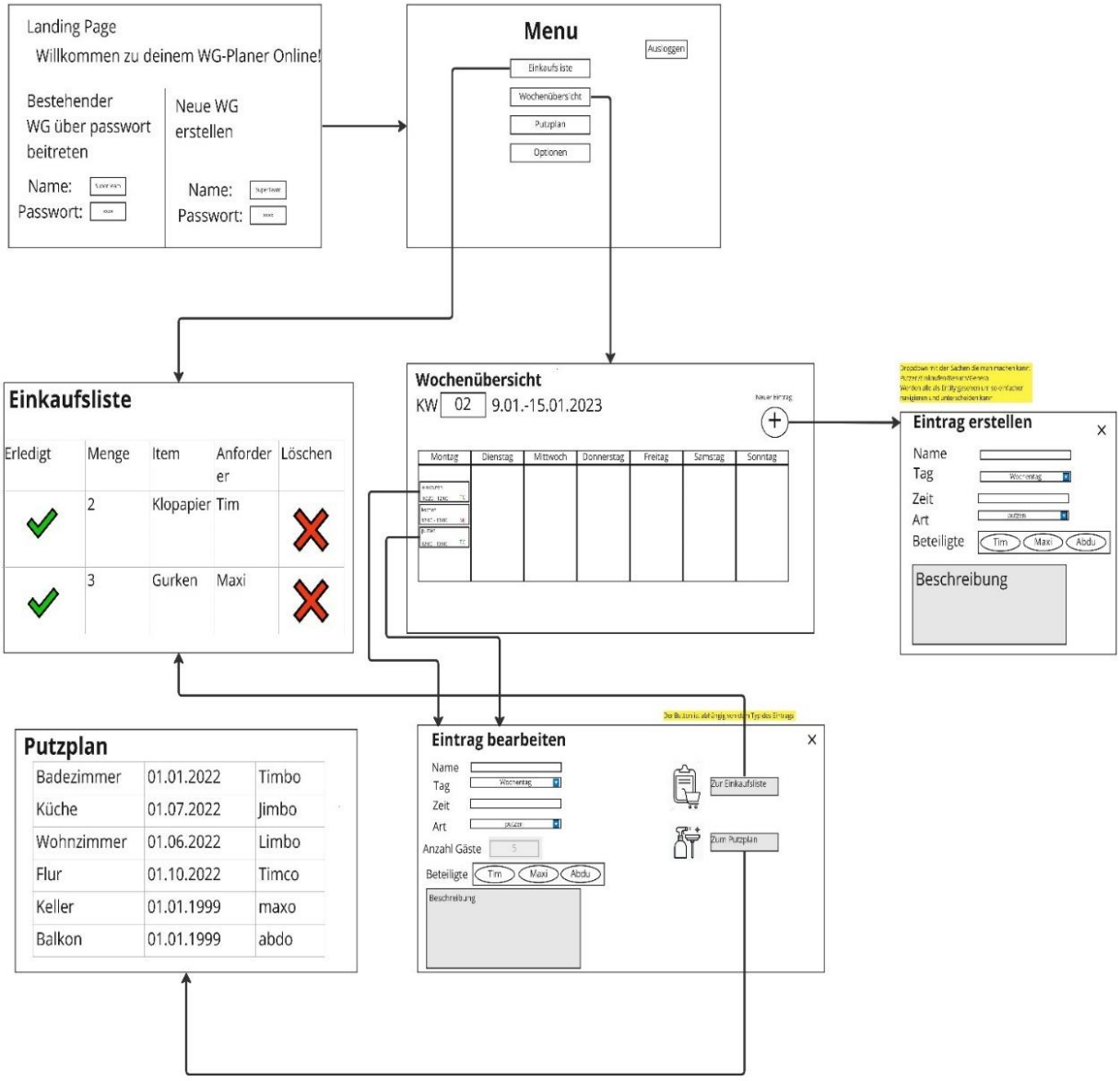


Abbildung 1 - Mockup der WG-Verwaltung

1.2 Ziel der Ausarbeitung

Autor: Tim Cirksena

Ist es machbar eine Benutzerfreundliche Web-Anwendung mit Quarkus und Java umzusetzen? Gibt es die Möglichkeit diese mit Quarkus Websockets umzusetzen?
Fragen wie diese wurden sich anfangs gestellt. Damit der Erfolg der Umsetzung überprüft werden kann, wurden vor Beginn der Bearbeitung gewissenhaft Akzeptanzkriterien

angelegt. Diese werden im Laufe des Berichts erörtert und in der Zusammenfassung aufgegriffen und beantwortet.

1.2.1 Musskriterien

Autor: Tim Cirksena, Maximilian Jaesch

- Jeder Nutzer kann eine WG angelegen und für die WG einen Namen und ein Passwort definieren
- Jeder Nutzer, der über den WG-Namen und das Passwort verfügt gilt als WG-Mitbewohner
- WGs können von WG-Mitbewohnern verändert und gelöscht werden
- Als WG-Mitbewohner kann ich auf einen Kalender zugreifen, der über Websockets aktualisiert wird.
- Der Kalender ist als Kalenderwoche des Jahres aufgebaut. Einträge wie Putzen, Einkauf, Besuch und Generelles können einem Datum des Kalenders hinzugefügt werden
- Die Einträge im Kalender sind verlinkt mit einzelnen Views. Wenn der WG-Mitbewohner einen Eintrag anklickt, wie z.B. "Einkaufen" wird die Einkauflisten-View geöffnet bzw. wird der Link für den Zugriff auf die Ressource zurückgegeben, auf den per GET-Request zugegriffen werden kann
- WG-Mitbewohner können Einkaufslisten einsehen und editieren, sowie die Liste oder einzelne Einkaufsposten als fertig/gekauft deklarieren.
- Es ist eine verteilte Anwendung, jeder Mitbewohner soll Änderungen Anderer live sehen können
- Als WG-Mitbewohner möchte ich Besuch ankündigen können und die Anzahl der Besucher, das Anliegen und den Zeitraum angeben können
  [1]

1.2.2 Wunschkriterien

Autor: Tim Cirksena, Maximilian Jaesch

- Ein Putzplan soll einsehbar sein, indem alle Orte aufgelistet werden an den, auf die WG bezogen, geputzt werden kann. Diese werden mit “zuletzt geputzt am: dd.mm.yyyy” versehen
- WG-Party kann geplant werden
- Es soll eine View für Große Ausgabe geben, z.B.
  - wer Miete bezahlt hat
  - Sofa kaufen, man kann beisteuern

[1]

1.2.3 Abgrenzungskriterien

Autor: Tim Cirksena, Maximilian Jaesch

- Die Qualität der umgesetzten Software und die Einhaltung der Qualitätsmerkmale ist höher zu priorisieren als die Quantität angebotener Funktionalität
- Als Qualitätsmerkmale erreicht werden soll a) User- und Developer-Experience (u.a. angemessenes API-Design, Design-Dokumentation, Swagger-UI, Dev-Services), b) Korrektheit der angebotenen Funktionalität, c) IT-Security (Zugriff auf die WG nur über Benutzername + Passwort, Input-Validierung) und d) Resilienz (u.a. Fault-Tolerance, Metriken zur Monitoring des Laufzeitverhaltens)
- Es werden zwei Schnittstellen zur Interaktion bereitgestellt, konkret:
    - Mensch-Maschine Interkation über eine Web-App, damit Team-Mitglieder diese Online einfach und angemessen über den Browser nutzen können
    - Maschine-Maschine Interaktion über eine REST-API, damit andere Entwickler eigene Lösungen, bspw. mobile Anwendungen realisieren können, um mit der App einfach und angemessen über eigene Tools zu interagieren

[1]

1.3 Aufbau der Hausarbeit

Autor: Tim Cirksena

Die Hausarbeit ist in unterschiedliche Kapitel unterteilt. Die Einleitung gibt dem Leser ein gewisses Verständnis, wohin diese Ausarbeitung führt. In dem Kapitel Darstellung der Grundlagen werden, im Rahmen der Projektarbeit, alle verwendeten Technologien erklärt und grundlegend aufarbeitet, sodass auch unerfahrene Leser ein Verständnis entwickeln können. Das nächste Kapitel Umsetzung erklärt die im Kapitel Darstellung der Grundlagen aufgelisteten Themen in Bezug auf die Verwendung im Projekt. Zum Beispiel wird die Verwendung von Websockets genauer erklärt und wie sich diese im Projekt bemerkbar machen. Im Entwicklungs-Kapitel wird der Aufbau genauerer erörtert, mithilfe von einem Klassendiagramm ist dieser Aufbau leichter zugänglich.

2 Darstellung der Grundlagen

Autor: Tim Cirksena

In diesem Teil des Projektberichts wird auf die Grundlagen der verwendeten Technologien eingegangen. Es soll klargestellt werden wieso und wofür einzelne Technologien verwendet und implementiert wurden. Dem Leser wird somit nähergebracht, wie die Software entwickelt und unter welchen Grundlagen sie entstanden ist.

2.1 SOLID

Autor: Tim Cirksena

SOLID ist ein Akronym für die fünf wichtigsten Prinzipien des Objektorientierten Designs, erstellt wurden diese von Robert C. Martin. Diese Prinzipien des SOLID, können auf viele verschiedene Programmiersprachen angewendet werden. Der Grundgedanke des SOLIDs ist wie sich die Software verhält bei dem Aufrechterhalten und Erweitern, wenn das Projekt wächst und größer wird. Unter Verwendung dieser Prinzipien in der Praktik können Refactoring von Code, adaptive Softwareentwicklung und Vermeidung von Code Smells beigetragen werden. [2]

2.1.1 Single-Responsibility-Prinzip

Autor: Tim Cirksena

Dieses Prinzip besagt, dass eine Klasse nur einen Use-Case ausführen soll. Wenn Änderungen an dieser Klasse passieren, soll es nur wenige Auswirkungen auf andere Klassen haben. Dadurch wird die Änderung des Codes eingegrenzt und folgend daraus das Fehlerrisiko gesenkt.

Wenn sich nicht an dieses Prinzip gehalten wird, verursacht das schnell viele Abhängigkeiten und eine hohe Vernetzung. Ab einem bestimmten Zeitpunkt kann dies dann zur Unhandlichkeit führen.

In der WG-Verwaltung wurde zum Beispiel darauf geachtet, dass die Resource-Klassen jeweils nur Response oder TemplateInstance bearbeiten.

2.1.2 Open-Closed-Prinzip

Autor: Tim Cirksena

Das Open-Closed-Prinzip beschreibt die Idee das eine Klasse offen für Erweiterungen und verschlossen für Modifikationen sein soll. Das heißt dass das Verhalten einer Klasse

zwar erweitert werden darf, aber nicht verändert. Das hat den Grund das, wenn eine Klasse nur durch eine Änderung innerhalb der Klasse erweitert werden kann, die Gefahr größer ist, dass hierdurch Fehler entstehen können.

Die wichtigsten Bedingungen des Open-Closed-Prinzips sind

- Vererbungen
- Verwendung von Interfaces

Hierdurch lassen sich neue Funktionalitäten innerhalb der Software einfach hinzufügen, ohne bestehende Klassen verändern zu müssen.

Dies ist gegeben durch die Verwendung von dem Entity-Controller-Boundary Pattern, es werden auf der Entity-Ebene, im Controller sowie in der Boundary Interfaces eingesetzt.

2.1.3 Liskovsche Substitutionsprinzip

Autor: Tim Cirksena

Liskovsche Substitutionsprinzip oder auch das Ersetzbarkeit Prinzip genannt zielt darauf ab, dass ein Objekt einer Unterklasse anstelle eines Objekts der Basisklasse verwendet werden kann, ohne dass das Programm seine Korrektheit verliert. Das bedeutet, dass diese eine Instanz einer Unterklasse alle Funktionalitäten und auch Eigenschaften der Basisklasse haben muss. Nur so gilt diese Klasse als gültiger Ersatz und kann verwendet werden.

Aufgrund dessen das die Management-Klassen alle Interfaces implementieren und die gegebenen Methoden so überschreiben, dass das Management als Vertreter des Interfaces gilt ist das Ersetzbarkeits-Prinzip auch in der WG-Verwaltung gegeben.

2.1.4 Interface-Segregation-Prinzip

Autor: Tim Cirksena

Nachdem Interface-Segregation-Prinzip sollen Interfaces nur die Funktionen besitzen, die auch wirklich eng zusammengehören. Es ist zu vermeiden das Interfaces große Kopplungen haben, sodass ansonsten unabhängige Clients auch unabhängig bleiben. Das reduziert den Komplexitätsgrad und verbessert die Wartbarkeit und Lesbarkeit des Codes.

2.1.5 Dependency-Inversion-Prinzip

Autor: Tim Cirksena

Das Dependency-Inversion-Prinzip besagt, dass höhere Abstrakte Klassen nicht von niedrigen abstrakten Klassen abhängen sollten.

2.2 REST-API

Autor: Maximilian Jaesch

Eine REST-API ist eine Schnittstelle zwischen Computern, die auf den Architekturstil REST, also „representational state transfer“ basiert. Über eine REST-API bieten Server Zugriff auf ihre Daten an, welche z.B. eine Datenbank aller bekannten Cocktailrezepte ist. Es werden von Clients über das Internet HTTP-Anfragen an Server verschickt, und der Server führt, je nach dem HTTP-Verb und den Parametern der Anfrage, bestimmte Operationen aus und beantwortet die Anfrage.

- GET: Abfragen von Daten aus der Datenbank
- POST: Erstellen von Daten in der Datenbank
- PUT/PATCH: Modifizieren von Daten in der Datenbank
- DELETE: Löschen von Daten in der Datenbank

Die Antworten des REST-Servers haben verschiedene Status-Codes, die abhängig von der Anfrage sind und teilen dem Client wichtige Informationen mit.

- 2XX: erfolgreiche Anfrage
- 4XX: Fehler bei der Anfrage des Clients
- 5XX: Fehler auf der Seite des Servers

REST-Apis kommunizieren mit JSON-Dateien, da diese für Computer leicht zu verarbeiten sind und gleichzeitig gut von Menschen lesbar sind. [3]

2.3 Server Side Rendering

Autor: Tim Cirksena

Server Side Rendering (SSR) ist der Prozess, bei dem eine Anwendung auf dem Server gerendert und dann an den Client gesendet wird. Dies passiert, bevor sie im Browser ausgeführt wird. Der Grund dahinter ist, dass durch das rendert im Vorhinein seitens des Servers, eine bessere Benutzerfreundlichkeit gegeben wird. Durch das schnellere Anzeigen auf den Endgeräten mit geringer Leistung.

Mit Quarkus kann Server Side Rendering einfach realisiert werden, da es bereits eine Umsetzung mit einem HTML-Template-System gibt, dieses nennt sich Qute.

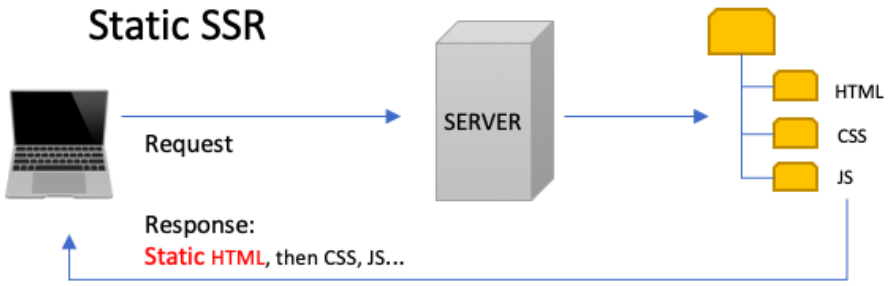


Abbildung 2 - Statisches SSR

//https://medium.com/tiny-code-lessons/client-side-static-and-server-side-rendering-e2769c381c09

In der WG-Verwaltung wird Static SSR nur im Login Bereich verwendet, weil hier keine dynamischen Informationen gebraucht werden.

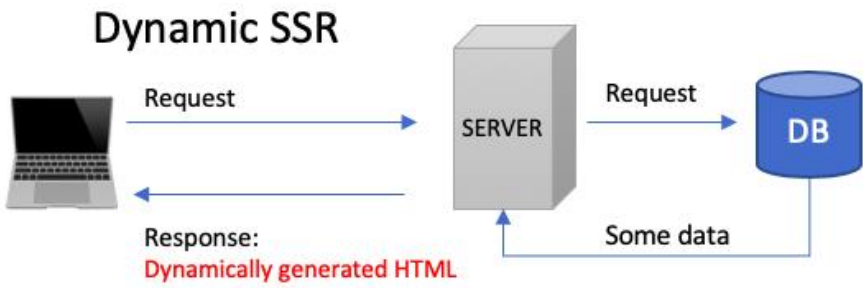


Abbildung 3 - Dynamisches SSR

///https://medium.com/tiny-code-lessons/client-side-static-and-server-side-rendering-e2769c381c09

Dynamic SSR wird in allen anderen Bereichen der WG-Verwaltung verwendet. Zum initialen laden des Kalenders, sowie bei der Einkaufsliste. Mittels Quarkus Qute werden Informationen dynamisch gerendert. Zu diesen Informationen zählen Einträge im Kalender und auch die aktuelle WG-Bezeichnung. Das Anzeigen von neu erstellten Einträgen ist ohne neu laden der Webseite nicht mittels Qute umsetzbar, aus diesem Grund werden hierfür Websockets verwendet. Diese Websockets sind nicht mehr dynamisches SSR.

2.4 Quarkus

Autor: Tim Cirksena

Quarkus ist ein Java-Framework, das von RedHat als Open-Source-Projekt veröffentlicht wurde und dessen Hauptaufgabe auf Kubernetes-basierten Containerplattformen liegt. Durch seine Konzipierung als Full-Stack-Lösung wurde das Framework so entwickelt, dass Java-Anwendungen optimal für diese Art von Plattformen laufen. Unter anderem sind Branchengrößen wie Microsoft Azure, RedHat OpenShift und Amazon AWS unterstützte Plattformen, aber auch zahlreiche andere, weniger bekannte Plattformen. Die Ersteller von Quarkus werben mit dem Slogan

- (Supersonic) schnelle Anlaufzeiten
- (Subatomic) wenig Speicherverbrauch

Das erreicht Quarkus indem versucht wird so viel wie möglich vom Anwendungscode schon in dem Buildvorgang zu verarbeiten. Das heißt dass nur Klassen, die auch wirklich zur Laufzeit benötigt werden, auch geladen werden.

Es gibt jedoch noch weitere Vorteile und Features die Quarkus, neben der Performanceverbesserung, mit sich bringt.

- **Modularität**: Quarkus ermöglicht es Entwicklern, nur die Funktionen und Abhängigkeiten in ihre Anwendung einzubinden, die auch wirklich benötigt werden. Dies führt dann unmittelbar zu kleineren, kompakteren und schnelleren Anwendungen.
- **Reaktivität**: Quarkus ist vollständig reaktiv, das bedeutet, dass die Anwendung auf asynchrone, nicht blockierende Weise arbeitet. Dadurch wird eine höhere Skalierbarkeit und Verfügbarkeit erreicht. Es wird nicht nur auf Anfragen gewartet, sondern auch auf Ereignisse reagiert. Hier unterscheidet sich Quarkus von

traditionellen, synchronen Anwendungen, die auf Anfragen warten müssen, bevor sie weiterarbeiten können.

- **Simple Integration**: Es wird eine einfache Integration mit anderen Technologien und Tools gegeben. Darunter zählen wie oben bereits erwähnt, Kubernetes, OpenShift, Red Hat Middleware, Java Virtual Machine (JVM) und mehr.

- **Extensions:** Quarkus bietet viele Bibliotheken in einer eigenen Variante an, diese werden als sogenannte Extensions bezeichnet. Der Code dieser Extensions wurde direkt auf die Zusammenarbeit mit Quarkus angepasst. Unteranderem zählen hierzu SmallRye OpenAPI und Apache Kafka. Insofern ein Projekt nach Quarkus importiert wird, können durch den Austausch dieser Bibliotheken zusätzlich Performance und Stabilität gewonnen werden.

Insgesamt bietet Quarkus eine leistungsstarke und flexible Lösung für die Entwicklung von Anwendungen für Kubernetes-basierte Containerplattformen. Mit den Vorteilen wie schnelle Startzeiten, geringem Speicherverbrauch, Reaktivität, einfacher Integration und zahlreichen Extensions, erleichtert es Entwicklern die Erstellung und Wartung ihrer Anwendungen. [4]

2.4.1 Qute

Autor: Tim Cirksena

Qute ist ein Feature von Quarkus welches dem Programmierer serverseitiges Templating erlaubt. Wie im oberen Teil: “Server Side Rendering” erläutert, geht es um den Prozess das die Anwendung seitens des Servers gerendert wird und erst dann zu dem Webbrowser gesendet wird. Damit wird die Performance verbessert, die Seiten laden schneller, und ein flüssigeres User-Interface wird ermöglicht. Außerdem kann Qute sehr einfach eingesetzt werden, aufgrund dessen, dass es als Teil des Quarkus-Frameworks bereits integriert ist. Es müssen deshalb keine weiteren Bibliotheken oder andere Abhängigkeiten heruntergeladen werden. Die Syntax von Qute ist außerdem sehr intuitiv gestaltet, durch HTML-Tags können auch Entwickler mit wenig oder keiner Erfahrung serverseitiges Rendering umsetzen. Qute unterstützt zusätzlich auch Funktionen wie die Verwendung von Schleifen und Variablen, so dass auch komplexere Anforderungen einfach umgesetzt werden können. [5] [6]

2.4.2 Simplified Hibernate ORM with Panache

Autor: Tim Cirksena

Panache ist eine Quarkus-spezifische Bibliothek, die es den Entwicklern vereinfacht, Datenbank-Operationen wie das Speichern, Abrufen und Aktualisieren von Daten aus einer Java Anwendung auszuführen. Die komplexen Details der Datenbank-Verbindung und der SQL-Abfragen werden seitens der Panache Bibliothek erledigt. Das heißt es müssen keine SQL-Abfragen geschrieben werden und es erleichtert den Übergang von der Quarkus Applikation und der Datenbank. [7]

2.4.3 Bean Validation API

Autor: Maximilian Jaesch

Die Bean Validation API stellt Annotationen zu Verfügung, die es erlauben Usereingaben bei der Persistierung von Panache Entitäten auf Korrektheit zu überprüfen. Es kann z.B. sichergestellt werden, dass Objekte nicht “null” sind und das Zahlen, die die Anzahl von einem Objekt darstellen sollen, nicht negativ sind. Wenn eine definierte Regel verletzt wird, wird eine “ConstraintViolationException” geworfen, die Abgefangen werden kann um dann dem REST-Client den Fehler-Code 422, der bei einer syntaktisch gültigen aber semantisch Fehlerhaften Entität zurückgegeben wird.

2.4.4 Testen der REST-API mit Swagger-UI

Autor: Maximilian Jaesch

Swagger-UI ist ein Bestandteil der SmallRye-OpenAPI Quarkus Bibliothek und erleichtert das manuelle Testen von REST-APIs mit einer praktischen Nutzeroberfläche. Mit Hilfe von Annotationen wie “@RequestBody” und “@ExampleObject” können vorgefertigte JSON-Dateien auf der Swagger-Oberfläche bereitgestellt werden, damit komplexe Objekte nicht bei jedem Test erneut eingegeben werden müssen.

2.5 Websockets

Autor: Tim Cirksena

Websockets bieten eine Lösung zur unkomplizierten und effizienten Kommunikation zwischen dem Server und dem Webbrowser. Die Websockets stellen

- Bidirektionale Kommunikation
- Vollduplexe Kommunikation
- Echtzeit-Client / Serverkommunikation

bereit und der Server kann somit jederzeit Daten an den Client senden.

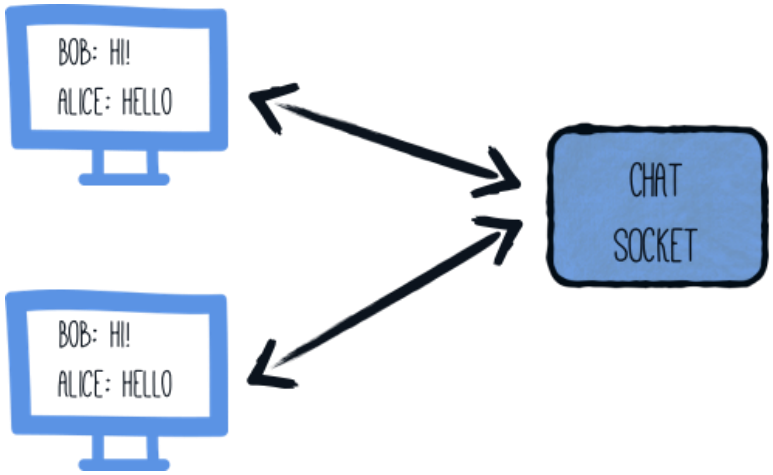


Abbildung 4 - Websocket Kommunikation

//https://quarkus.io/guides/websockets

Grundlegend werden vier Methoden verwendet, um auf Ereignisse zu reagieren, die bei der Verwendung von Websockets auftreten können. Durch diese folgenden vier Methoden “onOpen”, “onMessage”, “onClose” und “onError” können entsprechende Aktionen ausgeführt werden. [8]

2.5.1 OnOpen

Sobald eine Verbindung zwischen Client und Server hergestellt wurde, wird das “onOpen”-Ereignis von der Websocket Instanz ausgelöst. Es wird als initialer Handshake zwischen Client und Server bezeichnet. Dieses Ereignis, welches ausgelöst wird, sobald die Verbindung hergestellt ist, wird “onOpen” genannt.

2.5.2 OnMessage

Wenn der Server Daten sendet, tritt das “onMessage”-Ereignis auf. Die von dem Server an den Client gesendeten Nachrichten können Texte, Binärdaten oder Bilder enthalten. Jedes Mal, wenn Daten gesendet werden, wird die “onMessage”-Funktion ausgelöst.

2.5.3 OnClose

Wenn die Verbindung zwischen Client und Server besteht, und es nun zum Ende der Verbindung und somit auch zum Ende der Kommunikation kommt, wird das Close-Ereignis vom Websocket ausgelöst. Mit Hilfe des “onClose”-Ereignisses kann die Verbindung geschlossen werden. Nach dem Markieren des Endes der Kommunikation durch das “onClose”-Ereignis können keine weiteren Nachrichten mehr zwischen Server und Client übertragen werden.

2.5.4 OnError

Sofern das “onError”-Ereignis aufgerufen wird, gab es Fehler in der Verbindung. Das “onError”-Ereignis dient dazu auf diese Fehler aufmerksam gemacht zu werden. Dieses Ereignis endet auch immer mit dem Schließen der Verbindung und Kommunikation.

2.5.5 Websocket Aktionen

Einerseits gibt es die Ereignisse bei Websockets, wie oben schon benannt, werden diese ausgelöst, wenn etwas passiert. Andererseits werden Aktionen von Nutzern ausgeführt, wenn diese etwas bewirken wollen. Dieses Ausführen von Aktionen ist das Benutzen der Close()- und Send()-Funktionen.

2.5.6 Close() Funktion

Diese Methode dient dazu die Verbindung zu beenden, die Funktion ist als “goodbye handshake” zu sehen. Nachdem Aufruf können keine weiteren Daten verschickt werden, bis die Verbindung wieder hergestellt wurde.

2.5.7 Send() Funktion

Wenn eine offene Verbindung zwischen dem Client und Server besteht, kann der Client aktiv mit der Send()-Funktion Nachrichten an den Server senden. Sobald die Daten gesendet wurden, wird das “onMessage”-Ereignis ausgelöst, welches wiederum dem Server signalisiert, dass neue Daten empfangen wurden.

2.6 Bulma Framework

Autor: Tim Cirksena

Das Bulma Framework ist ein “Class-based” Framework für CSS und basiert auf dem Flexbox-Layout. “Class-based” ist so zu verstehen, das HTML-Elementen eine Klasse hinzugefügt wird und durch die Implementierung des Frameworks im HTML-Kopf, diese Elemente an das Bulma-Framework angepasst werden. Es werden somit, viele verschiedene CSS-Attribute dem Element zugewiesen und dies durch eine einfache Markierung. Das Framework hilft dabei Elemente einfach auszulegen und zu positionieren. [9] [10]

2.7 Entity Control Boundary Pattern

Autor: Tim Cirksena

Das Entity Control Boundary Pattern ist eine Art Software Architektur zu gestalten. Das Pattern ähnelt in einigen Punkten dem Model-View-Controller Pattern, einer dieser Punkte ist das beide Muster versuchen eine bessere Software zu ermöglichen, durch verbesserte Wartbarkeit und Skalierbarkeit. Im ECB-Pattern repräsentieren die Entitäten die Kern-Datenobjekte einer Software. Die Steuerung kümmert sich um die Business Logik und die Boundary definiert die Schnittstellen zu anderen Systemen. Das Pattern hat den Sinn die Verantwortung für die Verwaltung von Daten, die Steuerung von der Business Logik und den Zugriff auf externe Systeme zu trennen. Das ermöglicht dem Entwickler flexible und modulare Architektur zu entwickeln, die es vereinfacht Änderungen an der Software vorzunehmen, ohne dabei andere Teile der Anwendung zu beeinträchtigen. Ein weiterer Vorteil des ECB-Patterns ist die Möglichkeit unterschiedliche Typen von verschiedenen Entitäten zu verwalten. Diese können durch die Abgrenzungen des Patterns nebeneinander existieren, ohne voneinander abhängig zu sein. Grundlegend kann das ECB-Pattern eine praktische Lösung für die Architektur von einer Anwendung darstellen. Die Anwendung profitiert von effizienter Verwaltung von Daten, von Business Logik und Schnittstellen. [11]

2.8 Rest Assured Testing

Autor: Maximilian Jaesch

Es ist wichtig die erstellte Software auf Funktionalität und Richtigkeit zu testen. Dies geschieht mit manuellen Tests, welche mit der Swagger-UI und auch durch Ausprobieren der Funktionen auf der Webseite durchgeführt werden können. Da manuelles testen aber viel Zeit benötigt, wurden Bibliotheken entworfen, die automatisiert immer wieder dieselben Funktionalitäten überprüfen können. Solche Tests werden je nach Anzahl der geprüften Komponenten unterschiedlich genannt.

- Bei Unit-Tests werden einzelne Funktionen/Methoden eines Programms isoliert und auf die richtige Umsetzung ihrer vorgesehenen Funktionalität geprüft
- Bei Integration-Tests wird das Zusammenspiel mehrerer Software-Bausteine überprüft

Rest-Assured ist eine Bibliothek, die es ermöglicht Integration-Tests von REST-APIs automatisiert durchzuführen. Es können Anfragen an die REST-endpoints einer Applikation gesendet werden, um dann die Response Codes und den Response Body mit den erwarteten Werten abzugleichen

2.9 Docker

Autor: Tim Cirksena

Docker ist eine plattformübergreifende Open-Source-Container Lösung, sie ermöglicht es Entwicklern ihre Anwendung in isolierten Umgebungen zu erstellen und auszuführen. Container sind eine Art Virtuelle Maschine, die es ermöglichen Anwendungen auf einer

abgeschotteten Umgebung auszuführen. Dabei verwaltet Docker den Prozess der Erstellung und Verwaltung von Containern. Außerdem können die Anwendungen effektiv auf verschiedenen Plattformen und Umgebungen verteilt werden. Dadurch können Entwickler sicher gehen, dass ihre Anwendung auch auf anderen Umgebungen funktioniert. [12]

2.9.1 Was genau ist ein Docker-Container?

Ein sogenannter Docker-Container beschreibt eine Softwareeinheit, die alles bereitstellt, um eine Anwendung lauffähig zu machen. Darunter zählen verschiedene Komponenten:

- Programmcode
- RunTime-Engines
- Systemtools
- Systembibliotheken
- Einstellungen

Diese Komponenten werden auch als Softwarepaket bezeichnet. Das Softwarepaket läuft auf Linx-, Mac- und Windows-basierten Systemen. Der Container stellt sicher das die Software von der Umgebung isoliert ist und trotz Unterschiede einheitlich funktioniert.

2.10 Java.time für Verwaltung von Datum und Zeit

Autor: Maximilian Jaesch

Die Bibliothek java.time wurde in der JDK-Version 1.8 hinzugefügt [13]. Sie soll die Verwaltung von Kalender- und Zeitdaten standardisieren und vereinfachen. Es wurde diese Bibliothek für das Projekt ausgesucht, da sie Persistierung mit Panache und Serialisierung mit jsonb unterstützt, also mit den anderen verwendeten Technologien kompatibel ist.

2.11 Fullcalendar

Autor: Maximilian Jaesch

Fullcalendar ist eine beliebte Open-Source JavaScript Bibliothek [14], die einen visuell ansprechenden und mit JavaScript live editierbaren Kalender anbietet, der an das Design von Google Kalender errinnert. Für das Projekt sind die angebotenen Funktionalitäten der Wochenansicht, bei der die Zeiten in einem scrollbaren Interface dargestellt werden, und die statisch, und auch dynamisch, hinzufügbaren Events, die über eine bestimmte Zeit gehen und im Titel und ihrer Farbe angepasst werden können, besonders relevant.

3 Umsetzung

In diesem Kapitel werden die Grundlagen aufgegriffen und in Bezug auf die WG-Verwaltung drauf eingegangen. Es wird sich die Frage gestellt wie an einzelne Technologien rangegangen wurde und zum Nutzen der Software verwendet wurden.

3.1 Erstellen einer WG und Anmeldung

Autor: Maximilian Jaesch

Damit Nutzer eine WG anlegen können, wurde eine Landing Page mit Login und Register Funktion implementiert. Das Design der Seite wurde nach dem Prototyp erstellt. Hierfür wurde das Bulma-CSS Framework verwendet, vor allem die “columns” CSS-Klasse die HTML-Elemente in verschiedene Spalten einteilt und die “control” CSS-Klasse, die Input-Felder verschönert. [9]

Abbildung 5 - Landing Page der App

3.2 User Access Control

Autor: Maximilian Jaesch

Zur Authentifizierung haben einzelne WGs einen einzigartigen Namen und ein Passwort. Nutzer können sich mit diesem anmelden, um die Funktionalitäten der App zu nutzen.

Es wurde mithilfe von Quarkus Security eine Form-Based Authentication umgesetzt. Aufgrund der fehlenden Dokumentation auf den Quarkus Webseiten musste auf die Dokumentation von der Form-Based Authentication der Jakarta Servlets zurückgegriffen werden, weil Quarkus die Implementation davon übernommen hat.

Es muss ein Formular mit einer bestimmten Action und bestimmten IDs für Nutzername und Passwort verwendet werden, damit diese erkannt werden.

```
<form action="j_security_check" method=post>
    <p><strong>Please Enter Your User Name: </strong>
    <input type="text" name="j_username" size="25">
    <p><p><strong>Please Enter Your Password: </strong
        >
    <input type="password" size="15" name="j_password"
        >
    <p><p>
    <input type="submit" value="Submit">
    <input type="reset" value="Reset">
</form>

```

Snippet 1 - Ausschnitt aus index.html

Bei Login mit gültigen Nutzer-Daten wird ein Cookie gesetzt, welcher bei Logout wieder gelöscht wird.

Diese Art der Authentifizierung ist viel nutzerfreundlicher als Basic Authentication, da dort ein Logout über z.B. einen Button nicht möglich ist, und der Nutzer manuell seine Browserdaten für die Webseite aus dem Cache löschen muss.

Um das Ansprechen der gesicherten Endpoints der REST-Api mit Tools wie z.B. Insomnia zu ermöglichen, wurde die Basic Authentication neben der Form-Based Authentication auch aktiviert.

```
quarkus.http.auth.form.enabled=true
quarkus.http.auth.basic=true
```

Snippet 2 - Ausschnitt aus application.properties

Die Persistierung in der Datenbank wurde mit einer Panache-Entität und Annotationen der Quarkus Security Bibliothek umgesetzt.

3.3 REST-API

Autor: Maximilian Jaesch

In diesem Kapitel wird die umgesetzte REST-API der Applikation dargestellt. Diese wurde von der im Projektvorschlag angegebenen REST-API überarbeitet, da einige Pfade nicht sinnvoll waren.

/wgverwaltung	
GET	Alle WG-Namen mit ihren zugehörigen ID’s
PUT (angemeldet)	Namen und/oder Passwort der WG-Verwaltung ändern
POST	Anlegen einer neuen WG-Verwaltung – returns ID der neuen Verwaltung
DELETE	--

/wgverwaltung/{wgid}	
GET	Abfragen der WG-Verwaltung
PUT	--
POST	--
DELETE	WG-Verwaltung und alle untergeordneten Entitäten löschen
/wgverwaltung/kalender/{YYYY}/{KW} Nur angemeldete Nutzer	
GET	Kalendereinträge der bezüglichen Woche - zugehörig zur eigenen WG
PUT	--
POST	--
DELETE	--
/wgverwaltung/kalender Nur angemeldete Nutzer	
GET	--
PUT	--
POST	Anlegen eines Kalendereintrags
DELETE	--
/wgverwaltung/kalender/{kalenderEintragId} Nur angemeldete Nutzer	
GET	Abfragen eines Kalendereintrags
PUT	Ändern eines Kalendereintrags
POST	--
DELETE	Löschen eines Kalendereintrags
/wgverwaltung/einkaufsliste Nur angemeldete Nutzer	
GET	Abfragen der Einkaufsliste der WG
PUT	--
POST	Anlegen eines Eintrags in der Einkaufsliste
DELETE	--
/wgverwaltung/einkaufsliste/{einkaufslistenEintragId} Nur angemeldete Nutzer	
GET	Abfragen eines einzelnen Eintrags
PUT	Ändern eines einzelnen Eintrags
POST	--
DELETE	Löschen eines einzelnen Eintrags

3.3.1 Swagger-UI

Autor: Maximilian Jaesch

Es wurden mithilfe der in den Grundlagen genannten Annotationen verschiedene Beispiel-Objekte erstellt, um einen Kalender-Eintrag an die REST-API per Post Request zu senden.

```
@POST
@Transactional
@RequestBody(
        required = true,
        content = @Content(
                schema = @Schema(implementation = POSTKEintragDTO.class, required = true, requiredProperties = {"name"}),
                examples = {@ExampleObject(
                        name = "Example Value1",
                        description = "neuer kalendereintrag",
                        value = EXAMPLE_KALENDER_EINTRAG
```

Snippet 3 - Annotation um ein Beispielwert in die Swagger-UI einzufügen

Dies war notwendig aufgrund eines im Internet undokumentierten Fehlers, bei dem das von Swagger generierte Beispiel-Objekt ein falsches Format bei dem Datentyp “java.time.localtime” angibt.

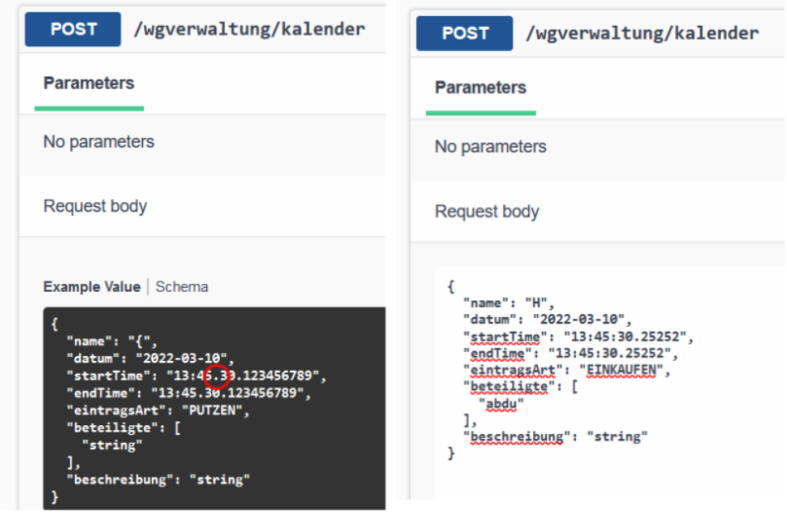


Abbildung 6 - Vergleich der fehlerhaften und Berichtigten JSON-Eingaben

Es wird zwischen der Minuten- und der Sekundenstelle anstatt von einem “:” ein “.” eingefügt, und dies führt zu einer “JsonbException” die den Grund des Fehlers nicht angibt.

3.3.2 ExceptionMapper für passende HTTP Response Codes

Autor: Maximilian Jaesch

Um geeignete HTTP Response Codes als Antwort auf die Client Requests abschicken zu können, werden bei Fehlern in der Applikation die geworfenen Exceptions von einer zentralen Klasse, der “ExceptionMappers” abgefangen. Die Bibliothek Jboss-RestEasy-Reactive stellt die Annotation “@ServerExceptionMapper” zur Verfügung, mit der man

Methoden annotieren kann, welche dann im Falle einer bestimmten Exception ausgeführt werden. In dieser Methode kann dann der gewünschte Response Code ausgesucht werden, und die Nachricht der geworfenen Exception in den Body der Response geschrieben werden, oder auch verworfen werden.

```
@ServerExceptionMapper
public RestResponse<String> mapException(NotFoundException e){
    JsonObject json = new JsonObject();
    json.put("message", "Not found" + e.getMessage());
    return RestResponse.status(RestResponse.Status.NOT_FOUND, json.toString());
}
```

Snippet 4 - Beispiel einer ExceptionMapper Funktion

Leider hat diese Annotation einen Fehler und kann bestimmte wichtige Exceptions wie “ConstraintViolationException” und “PersistenceException” nicht abfangen, was dann zu unleserlichen HTTP-Responses führt.

Es wurde ein Workaround gefunden, der die oben genannten Exceptions direkt bei der Quelle abfängt und eine für das Projekt neu erstelle Exception wirft. Diese Exception kann dann von dem Exception Mapper abgefangen werden, und ordentlich verarbeitet werden.

```
@Override
public ReturnKEintragDTO postKalenderEintrag(String wgName, POSTKEintragDTO dto) {
    try {
        KalenderEintrag kalenderEintrag = new KalenderEintrag(dto);
        kalenderEintrag.setWg(WG.find("wgname", wgName).firstResult());
        kalenderEintrag.persistAndFlush();
        return new ReturnKEintragDTO(kalenderEintrag);
    }catch (ConstraintViolationException c){
        throw new CustomConstraintViolationException(c.getMessage());
    }

}
```

Snippet 5 - Beispiel für Abfangen der ConstraintViolationException

3.4 Kalender View

Autor: Maximilian Jaesch

Um das Musskriterium zu erfüllen, einen Kalender mit Wochenansicht umzusetzen wurde die HTML-Seite mit der Fullcalendar-Bibliothek und dem Bulma CSS-Framework gestaltet. Das Design orientiert sich hier an dem, vor Beginn des Projektes erstellten, Mock-up.

Abbildung 7 - Wochenübersicht View

Der Nutzer kann die angezeigte Woche entweder auf die aktuelle Woche stellen, oder sich manuell eine Kalenderwoche in einem beliebigen Jahr aussuchen.

Des Weiteren wurde ein Modal erstellt, das mit Klick auf den „Eintrag hinzufügen“ Button geöffnet werden kann, und dem Nutzer die Möglichkeit bietet, einzelne Eigenschaften eines neuen Events festzulegen und dieses hinzuzufügen.

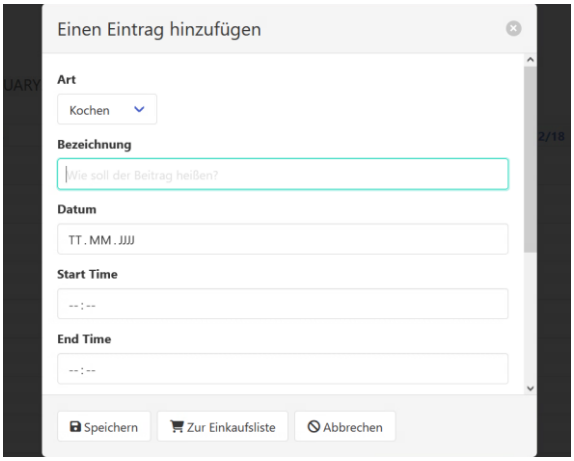


Abbildung 8 - Modal um einen Eintrag hinzuzufügen

Ein Ähnliches Modal wird mit Klick auf ein bestehendes Event geöffnet, was dem Nutzer erlaubt dieses Event zu bearbeiten, und per PATCH-Request zu aktualisieren und bei Bedarf wieder zu löschen.

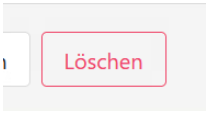


Abbildung 9 - Button der nur in dem bearbeitungs-Modal erscheint

3.5 Websockets in der WG-Verwaltung

Autor: Tim Cirksena

Websockets nehmen einen großen Teil der Funktionalität in der WG-Verwaltung ein. Sie dienen dazu das die WG-Mitbewohner die Änderungen anderer Mitbewohner in Echtzeit

mitbekommen. Die Änderungen werden an verschiedenen Stellen angezeigt. Im Hauptteil der Anwendung, dem Kalender, werden Kalendereinträge hinzugefügt. Sobald ein Eintrag abgeschickt wurde, bekommen alle anderen angemeldeten Sessions diese Änderungen angezeigt. Der Ablauf hierbei sieht wie folgt aus:

Der Client erstellt auf der Webseite per Ausfüllen des Modals einen neuen Kalendereintrag. Durch das Drücken des Speichern-Buttons wird der addEventListener ausgelöst in dem die Funktion für den POST-Request steckt.

```
const artInput = document.getElementById("dropdown-field");
const bezeichnungInput = document.getElementById("input-field-eintrag-name");
```

Snippet 6 - Der Input von den Inputfields wird temporär gespeichert

Es werden alle Inputs per ID geholt und somit die Usereingabe zwischen gespeichert. Die Eingaben werden dann an ein Objekt übergeben welches als DTO dient. In dem Beispiel des Kalenders ist es das POSTKEintragDTO. Dieses DTO wurde als Klasse in dem Anti-Corruption-Layer angelegt.

```
fetch("/wgverwaltung/kalender", {
    method: "POST",
    headers: {
        "Content-Type": "application/json"
    },
    body: JSON.stringify(obj)
```

Snippet 7 - fetch()-Funktion

Mithilfe der fetch()-Funktion kann das Objekt an die Ressource gesendet werden. Durch stringify wird das Objekt in JSON umgewandelt. Quarkus erkennt diese JSON und wandelt es beim Empfangen in das gewollte DTO um.

Nachdem alle Funktionen aufgerufen wurden, um den Kalendereintrag in die Datenbank zu schreiben, wird beim erfolgreichen Beenden der Methoden die Websocket Methode aufgerufen.

```
wochenWebsocket.wochenEintagCreate(fertigerEintrag);
```

Snippet 8 - Websocket wird Eintrag übergeben

Dies passiert im Management der jeweiligen Entität. Durch das Überreichen des fertigen Eintrags an den Websocket kann der Websocket nun das live-Update durchführen. In der Websocket Klasse wird das DTO nun in ein JsonB umgewandelt, das hat den Vorteil,

dass das DTO automatisch in eine JSON umgeschrieben wird und die Werte nicht einzeln übergeben werden müssen. JsonB wandelt ein JSON in eine binäre Darstellung um, welches für eine bessere Performance sorgen kann, jedoch wird diese Funktionalität in der WG-Verwaltung nicht verwendet. Das hat den Grund das bei diesem Datentyp nicht ohne weiteres ein Attribut hinzugefügt werden kann. Dieser Schritt ist essenziel für die Art und Weise in der bei der WG-Verwaltung Daten im Frontend, von einem Websocket, empfangen werden.

```
JsonObject typeHelper = new JsonObject(resultJson);
typeHelper.put("type", "wochenEintrag_created");
```

Snippet 9 - type wird eingefügt

Der Type “wochenEintrag_created” dient zur Identifikation des versendeten JSON.

```
broadcast(typeHelper.toString());
```

Snippet 10 - broadcast wird aufgerufen

Anschließend wird die Broadcast Methode des Websockets ausgeführt. Hier wird die JSON in einen String geschrieben und an alle angemeldeten Sessions geschickt.

```
private void broadcast(String json){
    for (Session session : sessions) {
        session.getAsyncRemote().sendText(json);
    }
}
```

Snippet 11 - broadcast Methode

Grundlegend dient die getAsyncRemote().sendText() Methode dazu ein JSON-Objekt in Form eines Strings an Websocket-Clients zu senden. Durch getAsyncRemote() wird der Remote-Endpunkt der Verbindung abgerufen, über den die Nachricht gesendet werden soll. Die Methode sendet das JSON-Objekt asynchron an den Websocket-Client, das heißt die anderweitige Ausführung der Anwendung wird nicht blockiert, bis die Übertragung abgeschlossen wurde. Der Websocket mit dem Remote-Endpunkt (/woche) empfängt nun per onmessage den gesendeten String.

```
socket.onmessage = function (event) {
    var message = JSON.parse(event.data);
```

Snippet 12 - Ausschnitt des Websocket Source Codes

Jetzt kann der Type verwendet werden, um die Message zuzuordnen.

```
if(message.type === "wochenEintrag_created"){
```

Snippet 13 - Type des Empfangenen Objekts wird überprüft

Die Message übergibt nun die notwendigen Parameter an eine Funktion in JavaScript, diese Funktionen sind relevant, um die Änderungen auch wirklich in Echtzeit anzeigen zu lassen. Sie erstellen ein temporäres HTML-Element mit den zu benötigenden Eigenschaften und Attributen.

3.6 Quarkus Qute

Autor: Maximilian Jaesch

Die Server-Side-Rendering Bibliothek Qute wurde für die Darstellung der Seiten Wochenübersicht und Einkaufsliste verwendet. Bei Aufruf der Seiten werden die jeweiligen Einträge, die zu der WG gehören aus der Datenbank geladen und direkt in die HTML-Seite eingefügt, bevor sie an den Client gesendet wird.

3.6.1 Template und Template Instance

Autor: Maximilian Jaesch

Endpoints für die HTML dateien sind genauso aufgebaut wie die REST-Endpoints und bedienen vorwiegend GET Requests. Der Unterschied besteht darin, dass der Rückgabewert eine sogenannte “TemplateInstance” ist, also eine HTML-Datei, die von Qute überarbeitet wurde.

```
@GET
@Transactional
@RolesAllowed({"admin","wg"})
@Retry(maxRetries = 4)
@Timeout(250)
@Operation(summary = "Einkauflisten-VIEW: Get all EinkaufslistenEinträge", description = "Returned alle E
public TemplateInstance getAllEintraeges(@Context SecurityContext securityContext){
    return einkaufslistenInterface.getAllEintraegeTemplate(securityContext.getUserPrincipal().getName());
}
```

Snippet 14 - Ausschnitt aus EinkaufslistenViewResource

Die „TemplateInstance“ wird mithilfe der „data()“ Methode auf einem mit „@Inject“ hinzugefügten Template erstellt. Dort können dann Key-Value Pairs von einer Bezeichnung und einem Java-Objekt übergeben werden, um diese dann in der HTML-Datei benutzen können.

```
@Override
public TemplateInstance getAllEintraegeTemplate(String wgName){
    return einkaufslisten_view.data("einkaufslisten", einkaufslistenCatalog.getAllEintraege(wgName));
}
```

Snippet 15 - Ausschnitt aus EinkaufslistenManagement

In der HTML-Datei kann dann über geschweifte Klammern “{ }“ ein Ausdruck markiert werden der von Qute bearbeitet werden soll.

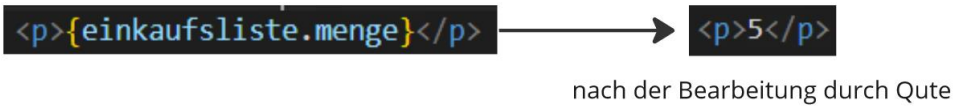


Abbildung 10 - Darstellung der Vorgehensweise von Qute

3.6.2 Schleifen in Qute

Autor: Maximilian Jaesch

Es können beim Aufrufen der “data()” Methode auch Listen übergeben werden, durch die auch mithilfe von Schleifen iteriert werden kann. Die HTML-Elemente, die sich innerhalb einer solchen Schleife befinden, werden für jeden Schleifendurchlauf erneut kopiert und mit den relevanten Daten aus dem übergebenen Java-Objekt befüllt.

```
{#for einkaufsliste in einkaufslisten}
    <div id="eintragbox{einkaufsliste.einkaufslistenEintragId}">
    <div class="columns is-vcentered">
    <div class="column is-narrow">···
    </div>
    <div class="column is-narrow">
        <p>{einkaufsliste.menge}</p>
    </div>
    <div class="column">
        <p>{einkaufsliste.item}</p>
    </div>
    <div class="column">
        <p>{einkaufsliste.besteller}</p>
    </div>
    <div class="column is-narrow">
    <button class="button is-danger" id="delete-eintrag"
    onclick="deleteEintrag({einkaufsliste.einkaufslistenEintragId})">
    </div>
    </div>
    </div>
{/for}
```

Snippet 16 - for-Schleife die in einkaufslisten_view.html die Einkaufsliste erstellt

Diese Funktionalität wird verwendet, um die einzelnen Einträge der Einkaufsliste und des Kalenders zu erstellen. Ein wichtiges Detail hierbei ist, dass auch JavaScript-Ausdrücke von Qute verändert werden können, welches essentiell ist, um beim Löschen eines Einkaufslisteneintrags das richtige Element auch in der Datenbank löschen zu können.

3.7 Input Validation mit HTML und Javascript

Autor: Maximilian Jaesch

Mit der Bean Validation API werden alle Entitäten, die in die Datenbank gespeichert werden, auf Richtigkeit überprüft. Wenn aber ein syntaktisch oder semantisch falsches Objekt übergeben wird, wird ein Fehler geworfen, der für den User der Webseite in einem Fehler-Popup angezeigt wird. Diese Art von Feedback kann zu einem frustrierendem Bedienerlebnis führen.

Um ein besseres Nutzererlebnis zu gewährleisten und die Bedienung zu erleichtern, können schon im Browser mit Hilfe der HTML Form Validation die Eingaben des Nutzers überprüft werden und für jedes einzelne Input Feld eigens angepasste Fehlermeldungen angezeigt werden.

Da die standardmäßige POST-Action von einem HTML-Formular nicht in JSON ist, wird eine angepasste JavaScript-Funktion verwendet, um die Daten des Input-Formulars herauszulesen, in ein JSON-Objekt umzuwandeln und per POST-Request abzuschicken. Dies hat den Nachteil, dass die HTML-Validation nicht automatisch ausgeführt wird und manuell aufgerufen werden muss.

Dieses Problem lässt sich mit der Constraint Validation API [15] lösen, die es ermöglicht, dass man die HTML-Validation einzelner Elemente und des ganzen Formulars dynamisch aufrufen kann.

```
const form = document.getElementById('modal-form');
if(!form.checkValidity()) {
    form.reportValidity();
```

Abbildung 11 - überprüfen der Validität und Anzeigen einer Fehlermeldung

Bevor die POST-Request an den Server geschickt wird, wird der Inhalt der einzelnen Felder überprüft und dem User eine sprechende Fehlermeldung angezeigt

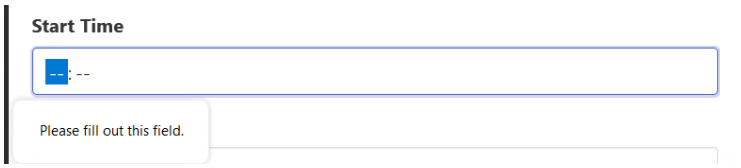


Abbildung 12 - Fehlermeldung in der Webseite

4 Entwicklung

4.1 ECB im Klassendiagramm

Autor: Maximilian Jaesch & Tim Cirksena

Es wird die genauere Entwicklung und Strukturierung der Anwendung durch eine Klassendiagramm designt.

4.1.1 Repository-Layer

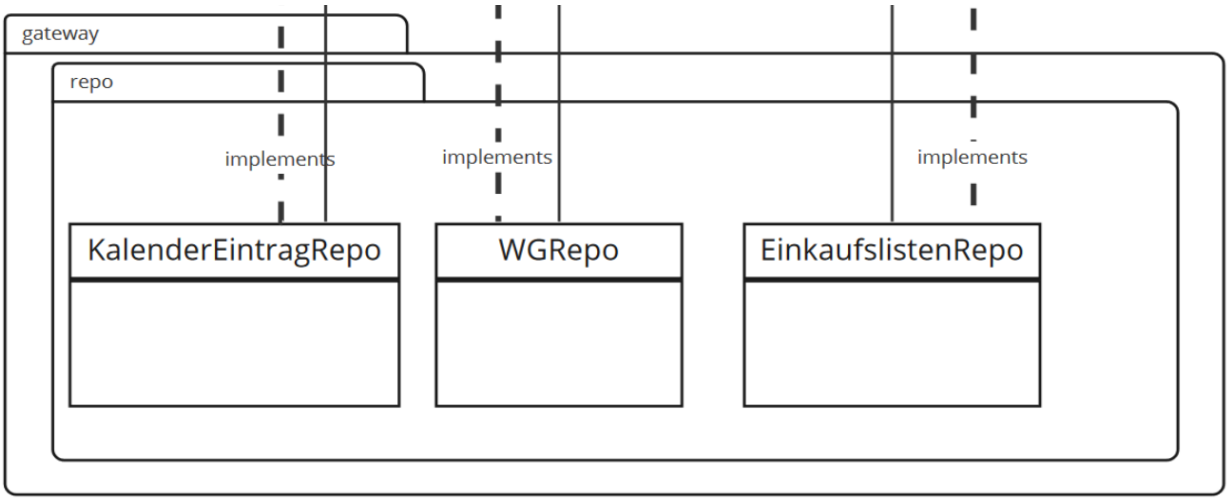


Abbildung 13 - Repository-Layer

Die Repositories implementieren den Katalog im Entität-Paket. Somit ist die Datenbank entkoppelt von der Business Logik.

4.1.2 Entity-Layer

Die Entitäten sind die Kern-Elemente der Datenstruktur, nur die Repository darf auf sie zugreifen und Veränderungen an ihnen vornehmen. Alle Anfragen, die über die Boundary kommen werden mittels DTOs erledigt.

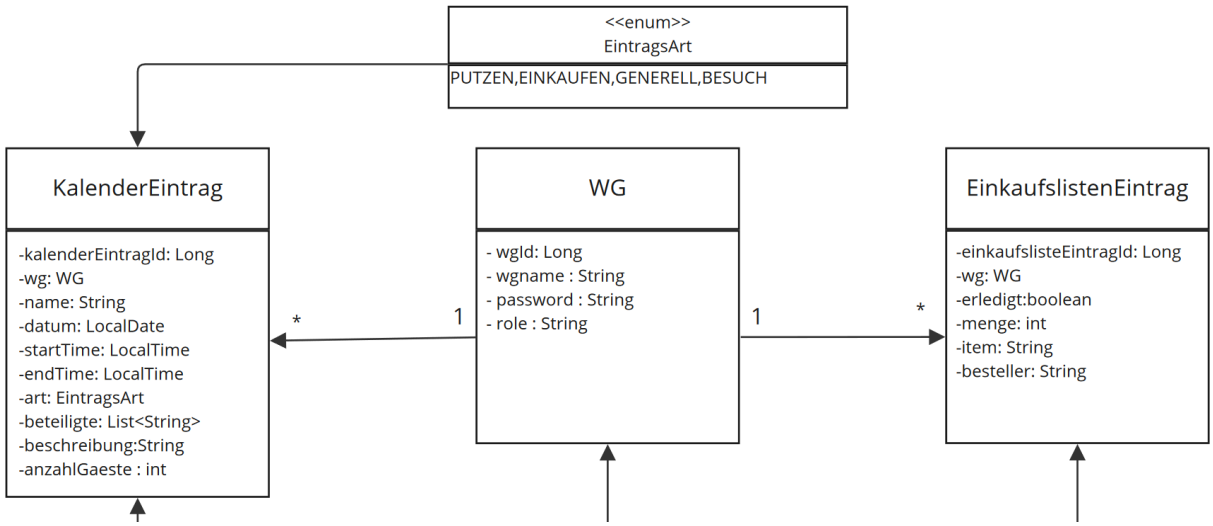


Abbildung 14 - Entity-Layer Entitäten

Die Kataloge halten alle Methoden die relevant sind für die CURD-Methoden und weitere Funktionen, die von der Repository und der Business Logik umgesetzt werden sollen. Die Kataloge werden von der Repository implementiert und verfügen somit über deren Methoden.

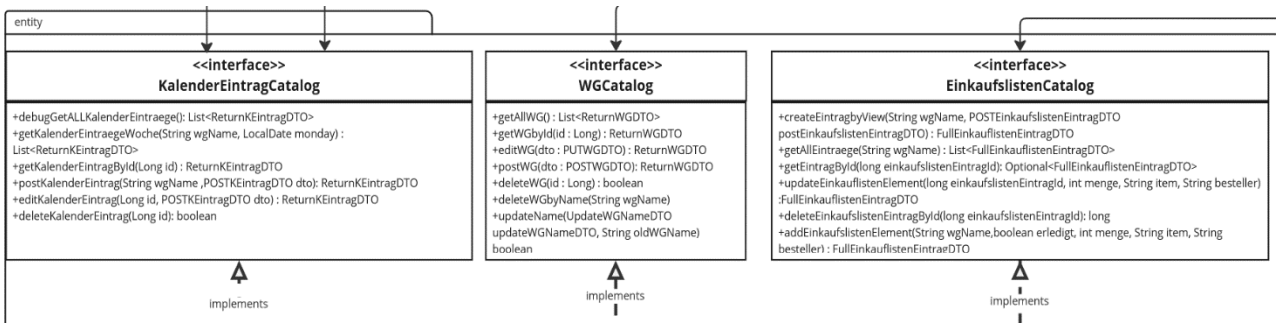


Abbildung 15 - Entity-Layer Catalogs

4.1.3 Control-Layer

Im Control-Layer wird die Business Logik umgesetzt, mithilfe von Services die Interfaces implementieren können. Diese können im Vorhinein die Responses für die Boundary fertigstellen. Durch diese Rangehens Weise erledigt das Control Layer wirklich die Business Logik und verarbeitet die Daten, die rein und raus gehen.

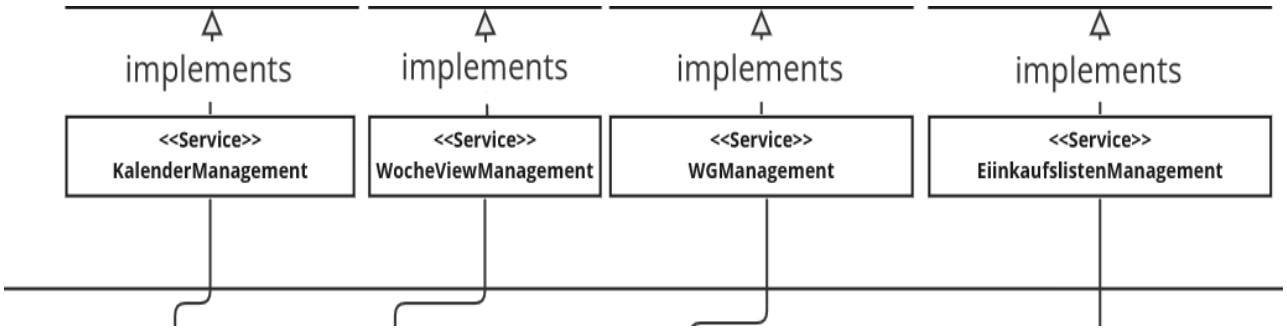


Abbildung 16 - Control-Layer Management

Die Service-Klassen halten einen Katalog damit die Kommunikation mit der Repository möglich ist.

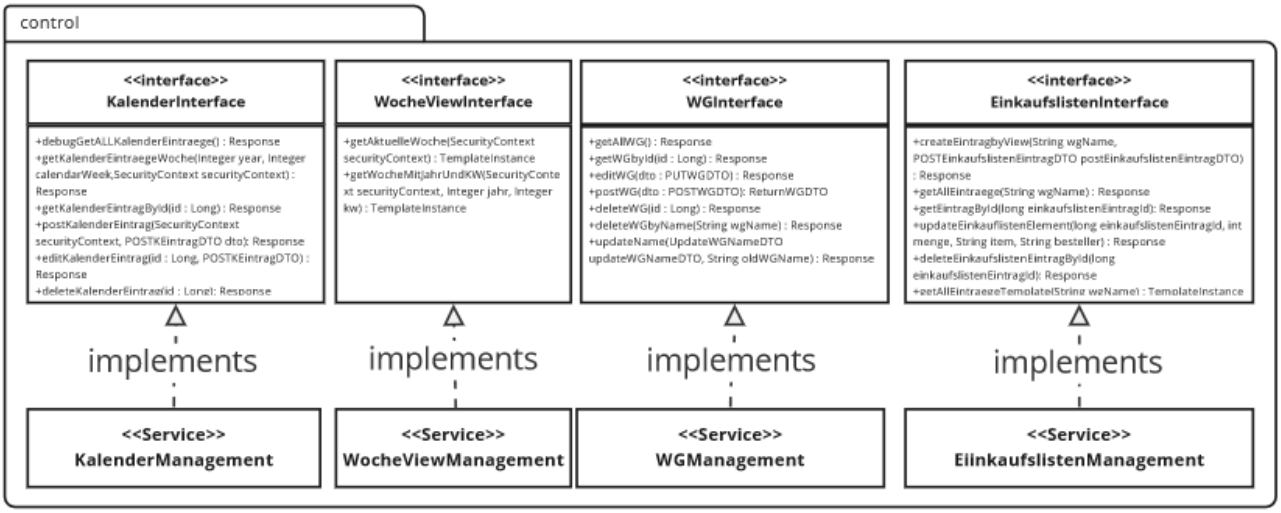


Abbildung 17 - Control-Layer Interfaces

4.1.4 Boundary-Layer Resource

Die Resources bekommen während der Laufzeit die Interfaces injected und können sofern eine Request kommt diese verarbeiten und an die Business Logik weitergeben.

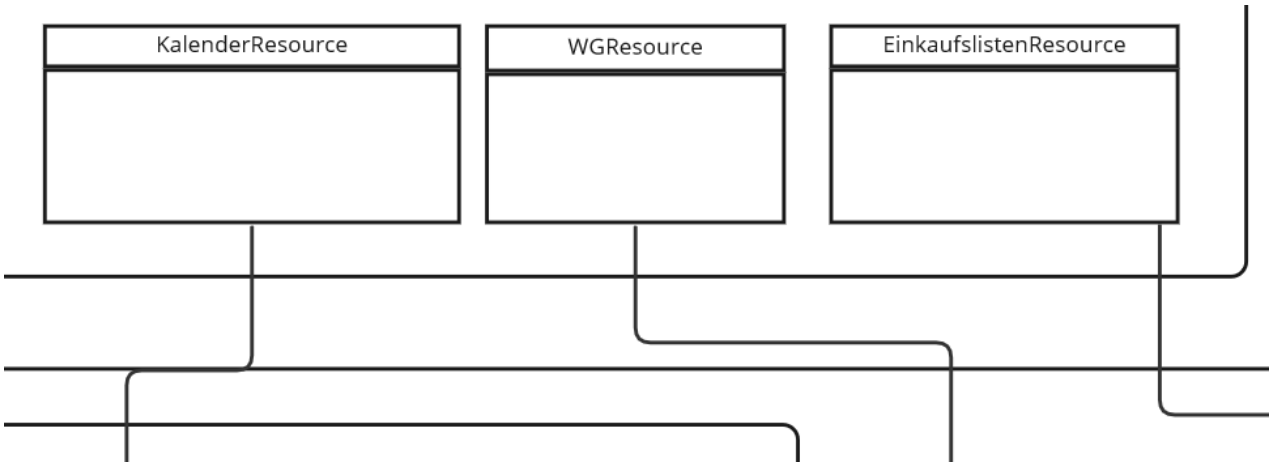


Abbildung 18 - Boundary-Layer Resource

Die View Resources dienen hauptsächlich dazu das spezifische Anfragen von der Webseite verarbeitet werden können, ebenfalls injecten diese die Interfaces der Business Logik, um somit die Anfragen weiterzugeben.

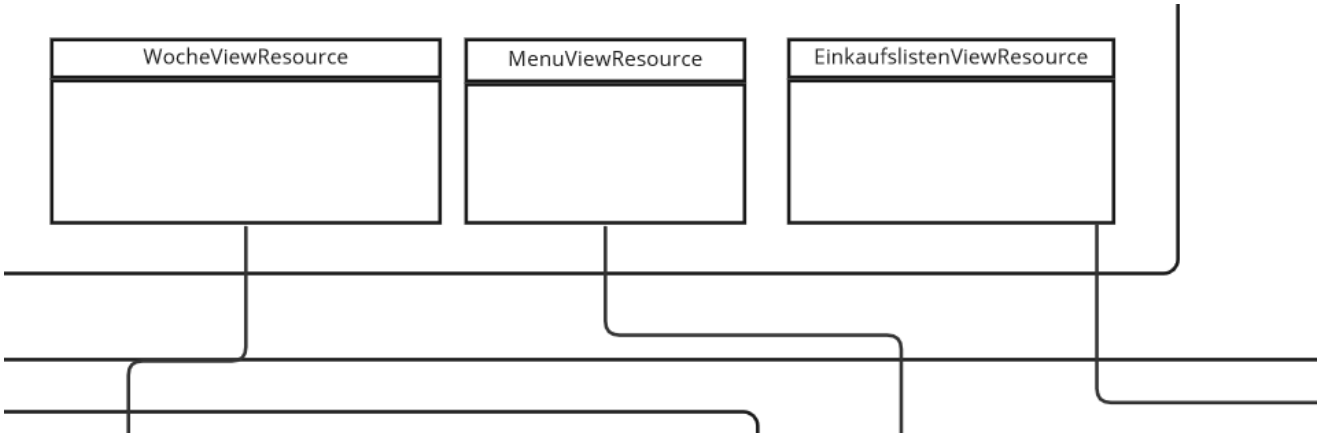


Abbildung 19 - Boundary-Layer Views

4.1.5 Boundary-Layer ACL

Das Anti-Corruption-Layer wird in der WG-Verwaltung genutzt, um die nötigen DTOs für Response-Verarbeitung zu kapseln. Die DTOs werden bis in die Repository übertragen, um dort Daten an die Entitäten zu übergeben oder von den Entitäten zu empfangen.

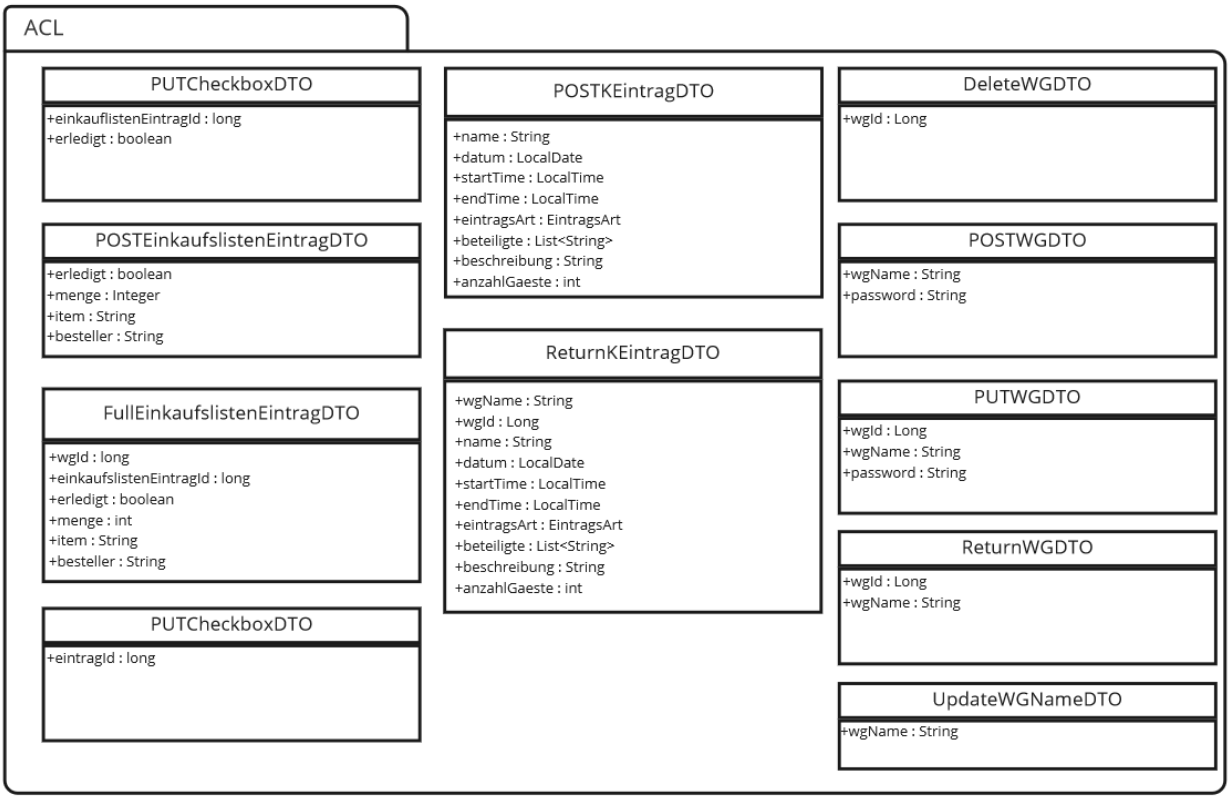


Abbildung 20 - Boundary-Layer ACLs

DTOs halten immer nur für den Zweck relevante Daten. Wenn zum Beispiel ein Name von der WG angepasst und geändert werden soll, wird nur der wgName als String an die Ressource überreicht.

4.1.6 Boundary-Layer Websockets

Das Websocket Package kümmert sich um die Verarbeitung von Anfragen und senden von Daten, die per Echtzeit update angezeigt werden müssen.

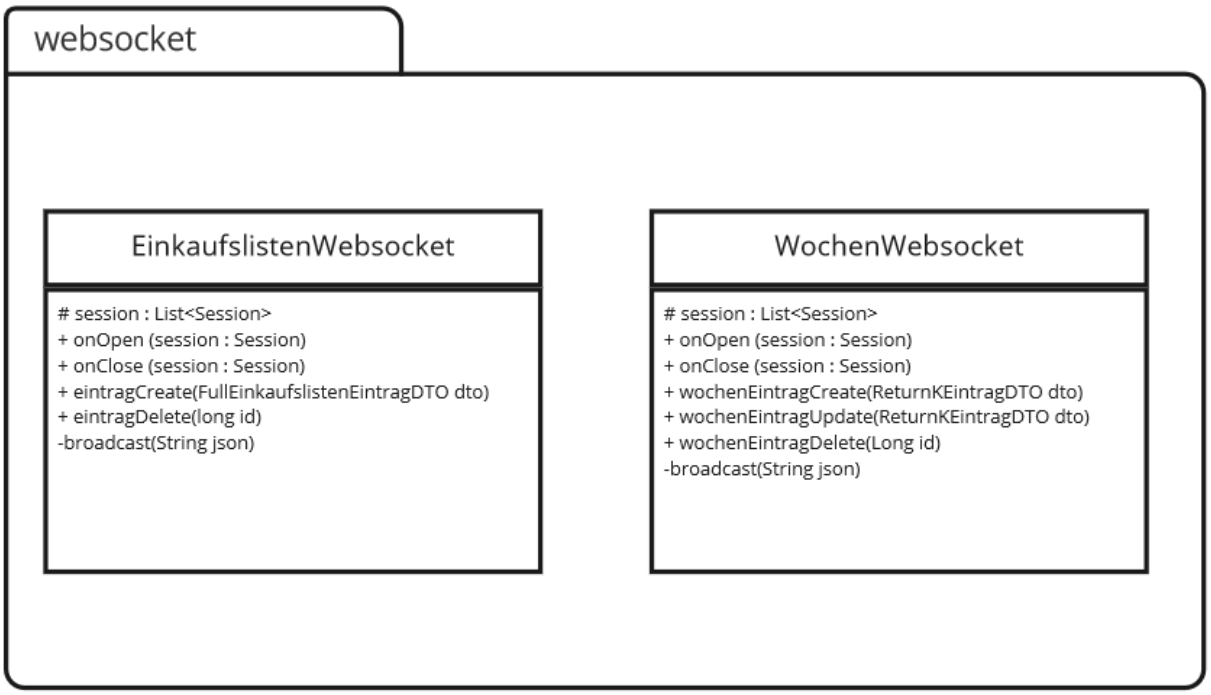


Abbildung 21 - Boundary-Layer Websockets

5 Zusammenfassung und Fazit

Autor: Maximilian Jaesch & Tim Cirksena

Rückblickend auf das Projekt und den Bericht wird eine Zusammenfassung und ein Fazit verfasst, was alles erreicht wurde und welches widerspiegeln soll was die Studierenden lernen konnten.

5.1 Zusammenfassung

Es wurde eine Web-App und eine REST-API erstellt, die den Nutzern hilft, ihre WG zu verwalten. Die Umsetzung erfolgte mit dem Technologie-Stack von Quarkus, welches viele Funktionalitäten bietet. Außerdem wurden Websockets für eine Echtzeit Lösung eingebunden, um die Nutzererfahrung reibungsloser zu gestalten. Durch Server Side Rendering, mit der Quarkus Erweiterung Qute, ist eine solide Performance gewährleistet. Als extra wurde das Wunsch Kriterium, WG-Party planen, umgesetzt, dies wurde ermöglich durch eine dynamische Implementierung des Kalenders, die es ermöglicht, bei Bedarf Anzahl von Gästen hinzuzufügen.

5.2 Fazit

Die Umsetzung des Projekts ist gelungen, da alle Muss Kriterien erfüllt wurden. Die Wunschkriterien wurden nicht alle umgesetzt, um sich auf die Qualität der Funktionalitäten der Muss Kriterien zu fokussieren. Für die Studenten war es ein großer Lernerfolg, da ihnen vor diesem Projekt die Technologien Websockets, das Bulma CSS-Framework und die Fullcalendar Bibliothek eher unbekannt waren und sie diese noch nicht umgesetzt haben. Des Weiteren haben sie die Datentypen, die sich mit Datum und Zeit befassen noch nie in einem Projekt verwendet und konnten deshalb neue Erkenntnisse sammeln. Außerdem wurde der Umgang mit Quarkus durch intensives Arbeiten verinnerlicht und Pattern wie das ECB-Pattern können nun problemlos angewendet werden.

6 Literaturverzeichnis

[1] R. Roosmann, *Prof. Dr.-Ing,* 2023.

[2] F. Listing, „microconsult.de,“ 21 05 2019. [Online]. Available: https://www.microconsult.de/blog/2019/05/fl_solid-prinzipien/. [Zugriff am 13 02 2023].

[3] A. S. Gillis, „techtarget.de,“ 01 09 2020. [Online]. Available: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API. [Zugriff am 14 02 2023].

[4] T. Neumann, „doubleslash.de,“ 19 11 2021. [Online]. Available: https://blog.doubleslash.de/quarkus-einfach-erklaert/. [Zugriff am 14 02 2023].

[5] G. Morling, „morling.dev,“ 03 01 2020. [Online]. Available: https://www.morling.dev/blog/quarkus-qute-test-ride/. [Zugriff am 15 02 2023].

[6] Quarkus.io, „Quarkus.io,“ 2023. [Online]. Available: https://quarkus.io/guides/qute. [Zugriff am 15 02 2023].

[7] T. Janssen, „thorben-janssen.com,“ [Online]. Available: https://thorben-janssen.com/introduction-panache/. [Zugriff am 14 02 2023].

[8] tutorialspoint, „Tutorialspoint,“ [Online]. Available: https://www.tutorialspoint.com/websockets/websockets_events_actions.htm#. [Zugriff am 12 02 2023].

[9] bulma, „bulma.io,“ [Online]. Available: https://bulma.io/. [Zugriff am 13 02 2023].

[10] J. Juviler, „hubspot.com,“ 7 12 2022. [Online]. Available: https://blog.hubspot.com/website/bulma-css. [Zugriff am 13 02 2023].

[11] cs.sjue.edu, 2020. [Online]. Available: https://www.cs.sjsu.edu/~pearce/modules/patterns/enterprise/ecb/ecb.htm. [Zugriff am 13 02 2023].

[12] opc-router, „opc-router.de,“ [Online]. Available: https://www.opc-router.de/was-ist-docker/. [Zugriff am 15 02 2023].

[13] „Package java.time,“ [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html.

[14] „The Most Popular JavaScript Calendar,“ FullCalendar LLC, [Online]. Available: https://fullcalendar.io/. [Zugriff am 11 Februar 2023].

[15] „Client-side form validation,“ [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation. [Zugriff am 13 02 2023].

[16] J. Gonzalez, „Medium.com,“ 16 12 2020. [Online]. Available: https://medium.com/swlh/how-to-build-a-websocket-applications-using-java-486b3e394139. [Zugriff am 12 02 2023].

Eidesstattliche Erklärung

Hiermit erkläre ich/ erklären wir an Eides statt, dass ich / wir die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt habe / haben. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche einzeln kenntlich gemacht. Es wurden keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Osnabrück, 17.02.2023 ..... M. Jarsch Tim Cirksena .....

Ort, Datum Unterschrift

(bei Gruppenarbeit die Unterschriften sämtlicher Gruppenmitglieder)

Urheberrechtliche Einwilligungserklärung

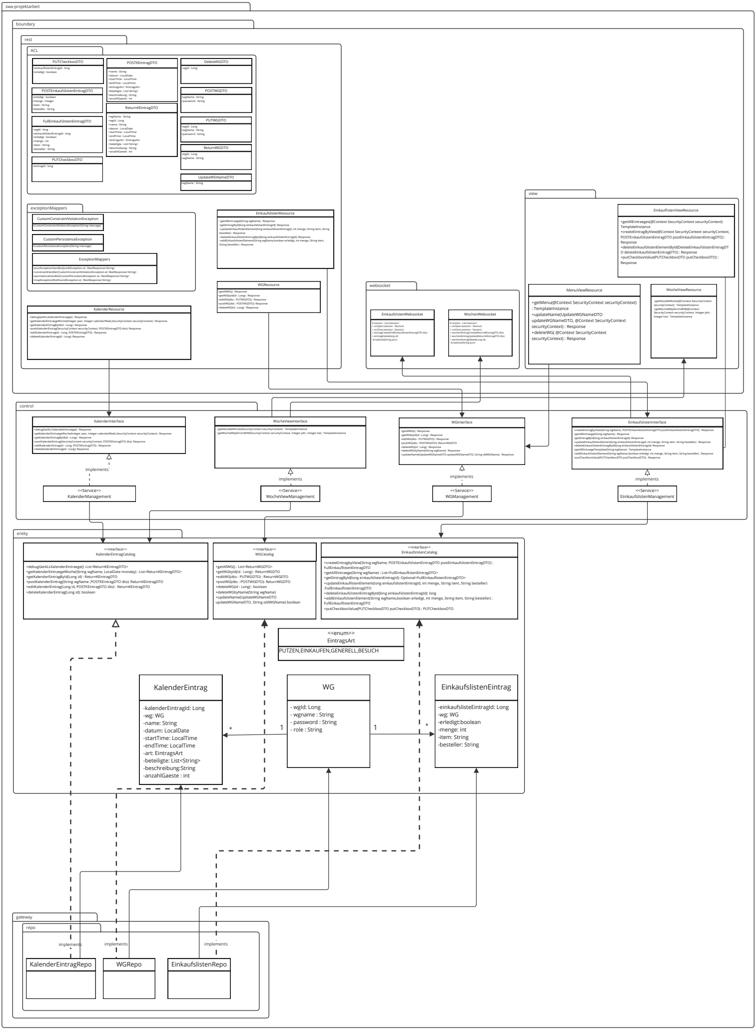
Hiermit erkläre ich/ Hiermit erklären wir, dass ich/wir damit einverstanden bin/sind, dass meine/ unsere Arbeit zum Zwecke des Plagiatsschutzes bei der Fa. Ephorus BV bis zu 5 Jahren in einer Datenbank für die Hochschule Osnabrück archiviert werden kann. Diese Einwilligung kann jederzeit widerrufen werden.

....

Ort, Datum Unterschrift

(bei Gruppenarbeit die Unterschriften sämtlicher Gruppenmitglieder)

Hinweis: Die urheberrechtliche Einwilligungserklärung ist freiwillig.



swa-projektarbeit
boundary
rest
ACL
PUTCheckboxDTO
POSTKEintragDTO
DeleteWGDTO
POSTEinkaufslistenEintragDTO
POSTWGDTO
FullEinkaufslistenEintragDTO
PUTWGDTO
ReturnKEintragDTO
ReturnWGDTO
PUTCheckboxDTO
UpdateWGNameDTO
exceptionMappers
CustomConstraintViolationException
CustomPersistenceException
ExceptionMappers
EinkaufslistenResource
WGResource
KalenderResource
view
EinkaufslistenViewResource
+getAllEintraeges(@Context SecurityContext securityContext): TemplateInstance
+createEintragByView(@Context SecurityContext securityContext, POSTEinkaufslistenEintragDTO postEinkaufslistenEintragDTO) : Response
+deleteEinkaufslistenElementById(DeleteEinkaufslistenEintragDTO deleteEinkaufslistenEintragDTO) : Response
+putCheckboxValue(PUTCheckboxDTO putCheckboxDTO) : Response
websocket
EinkaufslistenWebsocket
WochenWebsocket
MenuViewResource
+getMenu(@Context SecurityContext securityContext) : TemplateInstance
+updateName(UpdateWGNameDTO updateWGNameDTO, @Context SecurityContext securityContext) : Response
+deleteWG(@Context SecurityContext securityContext) : Response
WocheViewResource
control
KalenderInterface
WocheViewInterface
WGInterface
EinkaufslistenInterface
implements
<<Service>>
KalenderManagement
WocheViewManagement
WGManagement
EiinkaufslistenManagement
entity
<<interface>>
KalenderEintragCatalog
+debugGetALLKalenderEintraege(): List<ReturnKEintragDTO>
+getKalenderEintraegeWoche(String wgName, LocalDate monday) : List<ReturnKEintragDTO>
+getKalenderEintragById(Long id) : ReturnKEintragDTO
+postKalenderEintrag(String wgName ,POSTKEintragDTO dto): ReturnKEintragDTO
+editKalenderEintrag(Long id, POSTKEintragDTO dto) : ReturnKEintragDTO
+deleteKalenderEintrag(Long id): boolean
<<interface>>
WGCatalog
+getAllWG() : List<ReturnWGDTO>
+getWGbyId(id : Long) : ReturnWGDTO
+editWG(dto : PUTWGDTO) : ReturnWGDTO
+postWG(dto : POSTWGDTO): ReturnWGDTO
+deleteWG(id : Long) : boolean
+deleteWGbyName(String wgName)
+updateName(UpdateWGNameDTO updateWGNameDTO, String oldWGName) boolean
<<interface>>
EinkaufslistenCatalog
+createEintragbyView(String wgName, POSTEinkaufslistenEintragDTO postEinkaufslistenEintragDTO) : FullEinkaufllistenEintragDTO
+getAllEintraege(String wgName) : List<FullEinkaufllistenEintragDTO>
+getEintragById(long einkaufslistenEintragId): Optional<FullEinkaufllistenEintragDTO>
+updateEinkaufslistenElement(long einkaufslistenEintragId, int menge, String item, String besteller) :FullEinkaufllistenEintragDTO
+deleteEinkaufslistenEintragById(long einkaufslistenEintragId): long
+addEinkaufslistenElement(String wgName,boolean erledigt, int menge, String item, String besteller) : FullEinkaufllistenEintragDTO
+putCheckboxValue(PUTCheckboxDTO putCheckboxDTO) : PUTCheckboxDTO
<<enum>>
EintragsArt
PUTZEN,EINKAUFEN,GENERELL,BESUCH
KalenderEintrag
-kalenderEintragId: Long
-wg: WG
-name: String
-datum: LocalDate
-startTime: LocalTime
-endTime: LocalTime
-art: EintragsArt
-beteiligte: List<String>
-beschreibung:String
-anzahlGaeste : int
WG
- wgId: Long
- wgname : String
- password : String
- role : String
EinkaufslistenEintrag
-einkaufslisteEintragId: Long
-wg: WG
-erledigt:boolean
-menge: int
-item: String
-besteller: String
*
1
1
*
gateway
repo
implements
implements
implements
KalenderEintragRepo
WGRepo
EinkaufslistenRepo