

# Raumbelegungssoftware über Samply.Beam

Abschlussprüfung Sommer 2024 Fachinformatiker für Anwendungsentwicklung

Martin Jurk  
Beethovenstraße 8  
68753 Waghäusel  
Prüflingsnummer:  
3412

**Ausbildungsbetrieb:**

Deutsches Krebsforschungszentrum  
im Neuenheimer Feld 280  
69120 Heidelberg

## Inhaltsverzeichnis

|   |           |
|---|-----------|
| <b>Erklärung</b>                                      | <b>4</b>  |
| <b>Abbildverzeichnis</b>                              | <b>4</b>  |
| <b>Tabellenverzeichnis</b>                            | <b>5</b>  |
| <b>Glossar</b>  | <b>5</b>  |
| <b>1 Einleitung</b>                                   | <b>5</b>  |
| 1.1 Thema der Projektarbeit                           | 6         |
| 1.2 Betriebsumfeld                                    | 6         |
| 1.3 Projektbeschreibung                               | 6         |
| 1.4 Projektziel                                       | 7         |
| 1.5 Projektumfeld                                     | 7         |
| 1.6 Projektschnittstellen                             | 7         |
| <b>2 Projektplanung</b>                               | <b>8</b>  |
| 2.1 Erarbeitung der Anforderung                       | 8         |
| 2.2 Prüfung der Vorarbeiten                           | 8         |
| 2.3 Konzept Recherche                                 | 8         |
| 2.4 Personalplanung                                   | 9         |
| 2.5 Sachmittelplanung                                 | 9         |
| 2.6 Kostenplanung                                     | 9         |
| 2.7 Amortisationsrechnung                             | 10        |
| 2.8 Projektphasen- und Zeitplanung                    | 10        |
| 2.9 Abgleich der Anforderung mit Auftraggeber         | 11        |
| 2.9.1 Evaluation und Analyse Themen                   | 11        |
| <b>3 Analysephase</b>                                 | <b>11</b> |
| 3.1 Ist-Analyse                                       | 11        |
| 3.1.1 Standorte und Räume in der Datenbank            | 11        |
| 3.1.2 Datenbank Registrierung für Arbeitsplatz (Büro) | 11        |

|          |  |           |
|----------|--|-----------|
| 3.2      | <i>Konzepte</i> .....                                      | 13        |
| 3.2.1    | Variante A dezentrales Netzwerk (kleine Brückenköpfe)..... | 13        |
| 3.2.2    | Variante B mit mobiler App: .....                          | 13        |
| 3.2.3    | Tauri .....  | 13        |
| 3.2.4    | Variante C zentrale Client-Komponente: .....               | 13        |
| 3.3      | <i>Soll-Analyse</i> .....                                  | 13        |
| <b>4</b> | <b>Entwurfsphase</b> .....                                 | <b>14</b> |
| 4.1      | <i>Auswahl der Bibliotheken</i> .....                      | 14        |
| 4.2      | <i>Ermittlung des Funktionsumfang</i> .....                | 14        |
| 4.3      | <i>Deployment</i> .....                                    | 14        |
| 4.4      | <i>Entwurf Front - und Backend</i> .....                   | 14        |
| 4.5      | <i>Schnittstellen zu Samply.Beam</i> .....                 | 15        |
| <b>5</b> | <b>Implementierung</b> .....                               | <b>15</b> |
| 5.1      | <i>Lokale Testumgebung</i> .....                           | 15        |
| 5.2      | <i>Datenbankanbindung</i> .....                            | 15        |
| 5.3      | <i>Client Front- und Backend Komponente</i> .....          | 16        |
| 5.4      | <i>Samply.Beam Anbindung</i> .....                         | 16        |
| 5.4.1    | Client Samply.Beam .....                                   | 16        |
| 5.4.2    | BackendConnector Samply.Beam.....                          | 17        |
| <b>6</b> | <b>Projektabschluss</b> .....                              | <b>17</b> |
| 6.1      | <i>Soll-Ist-Vergleich</i> .....                            | 17        |
| 6.2      | <i>Übergabe</i> .....                                      | 18        |
| 6.3      | <i>Ausblick und Fazit</i> .....                            | 18        |
|          | <b>Literaturverzeichnis</b> .....                          | <b>18</b> |
| <b>7</b> | <b>Anlagen</b> .....                                       | <b>18</b> |
| 7.1      | <i>Broker Beispiel</i> .....                               | 19        |
| 7.2      | <i>Bridgehead</i> .....                                    | 19        |
| 7.3      | <i>Basis Beam Beispiel</i> .....                           | 20        |
| 7.4      | <i>Mrbs Datenbank</i> .....                                | 21        |
| 7.5      | <i>dezentral Netzwerk kleine Brückenköpfe</i> .....        | 22        |
| 7.6      | <i>Mobile App</i> .....                                    | 22        |

|          |   |           |
|----------|---|-----------|
| 7.7      | <i>Mobile App Tauri</i> .....   | 23        |
| 7.8      | <i>zentrale Client Komponente</i> .....                                       | 24        |
| 7.9      | <i>Nutzwert Analyse Frontend Client</i> .....                                 | 24        |
| 7.10     | <i>Zeitablauf des Projekts als Meilensteine</i> .....                         | 25        |
| 7.11     | <i>Übersicht für Raum und Standortbelegung</i> .....                          | 25        |
| 7.12     | <i>Entwurf Backend mit Datenbank</i> .....                                    | 26        |
| 7.13     | <i>Entwurf Darstellung Client Single Webpage</i> .....                        | 26        |
| 7.14     | <i>Komponenten Anbindung über Samply.Beam</i> .....                           | 27        |
| 7.15     | <i>Datenstruktur für die Tagesübersicht</i> .....                             | 28        |
| 7.16     | <i>Samply.Beam Task</i> .....   | 28        |
| 7.17     | <i>Samply.Beam Task Result</i> .....  | 29        |
| 7.18     | <i>Lokale Testumgebung Docker-Compose-Config</i> .....                        | 29        |
| 7.19     | <i>Funktion für Tagesansicht Backend</i> .....                                | 30        |
| 7.20     | <i>Funktion Arbeitsplatz buchen</i> .....                                     | 33        |
| 7.21     | <i>Funktion Arbeitsplatzbuchung löschen</i> .....                             | 33        |
| 7.22     | <i>Funktionen von BeamBackendConnector</i> .....                              | 34        |
| 7.23     | <i>Template angepasste Routen für den Client Back- und Frontend</i> .....     | 35        |
| 7.24     | <i>Funktionen des Clients um mit Beam zu kommunizieren</i> .....              | 36        |
| 7.25     | <i>Eigene Bibliothek für geteilte Datentypen</i> .....                        | 39        |
| 7.26     | <i>JavaScript Funktionen um mit dem Client Backend zu kommunizieren</i> ..... | 42        |
| 7.27     | <i>Svelte Singlepage mit Tagesansicht</i> .....                               | 44        |
| 7.28     | <i>Angepasste Funktionen von Samply.Beam</i> .....                            | 46        |
| 7.29     | <i>UML BackendConector und Client</i> .....                                   | 48        |
| 7.30     | <i>SharedTypes Bibliothek</i> .....   | 49        |
| 7.31     | <i>Projektauftrag</i> .....   | 49        |
|          | <i>Einleitung</i> .....   | 49        |
|          | <i>Aktueller Stand</i> .....  | 50        |
|          | <i>Notwendige Schritte</i> .....  | 50        |
| 7.32     | <i>Zeitplanung</i> .....  | 51        |
| <b>8</b> | <b>Kundendokumentation</b> .....  | <b>52</b> |
| 8.1      | <i>Room Booking System over Samply.Beam</i> .....                             | 52        |

|       |  |    |
|-------|--|----|
| 8.1.1 | Des cription "Raumbelegungssoftware" ..... | 52 |
| 8.1.2 | Technologiestack.....                      | 52 |
| 8.1.3 | Installation .....                         | 52 |

## Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit und deren Anlagen selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Darüber hinaus erkläre ich, dass diese Abschlussarbeit bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form, selbst auszugsweise, vorgelegt wurde. In dieser Dokumentation wurden blindenspezifische Hilfsmittel genutzt.

Heidelberg, 24.05.2024

Martin Jurk

## Abbildverzeichnis

|   |    |
|---|----|
| Abbildung 1: Broker Beispiel .....                      | 19 |
| Abbildung 2: Bridgehead .....                           | 20 |
| Abbildung 3: Basis Beam Beispiel .....                  | 20 |
| Abbildung 4: Mrbs Datenbank .....                       | 21 |
| Abbildung 5: kleine Brückenköpfe .....                  | 22 |
| Abbildung 6: mobile App Basis .....                     | 22 |
| Abbildung 7: Tauri App .....                            | 23 |
| Abbildung 8: zentrale Client Komponente .....           | 24 |
| Abbildung 9: Zeitablauf und Meilensteine .....          | 25 |
| Abbildung 10: Übersicht Raum und Standortbelegung ..... | 25 |
| Abbildung 11: Backend Entwurf .....                     | 26 |
| Abbildung 12: Darstellung Single Webpage Entwurf .....  | 26 |
| Abbildung 13: Anbindung über Samply.Beam .....          | 27 |
| Abbildung 14: Datenstruktur für Tagesübersicht .....    | 28 |
| Abbildung 15: Samply.Beam Task .....                    | 28 |
| Abbildung 16: Samply.Beam Task Result .....             | 29 |
| Abbildung 17: BackendConnector und Client UML .....     | 48 |
| Abbildung 18: SharedTypes Bibliothek .....              | 49 |

## Tabellenverzeichnis

|   |    |
|---|----|
| Tabelle 1: Projektumfeld .....                      | 7  |
| Tabelle 2: Nutzwertanalyse Programmiersprache ..... | 8  |
| Tabelle 3: Personalplanung .....                    | 9  |
| Tabelle 4: Sachmittelplanung Hardware.....          | 9  |
| Tabelle 5: Amortisationsrechnung .....              | 10 |
| Tabelle 6: Grober Zeitplan und Meilensteine .....   | 11 |
| Tabelle 7: Nutzwertanalyse Client .....             | 24 |
| Tabelle 8: Detaillierte Zeitplanung .....           | 52 |

## Glossar

| Abkürzung | Erklärung                        |
|-----------|----------------------------------|
| DKFZ      | Deutsches Krebsforschungszentrum |
| Beam      | Samply.Beam (Abteilungsprojekt)  |
| MRBS      | Meeting Room Booking System      |
| JSON      | JavaScript Objekt Notation       |
|           |                                  |
|           |                                  |
|           |                                  |
|           |                                  |
|           |                                  |

## 1 Einleitung

Dieses interne Projekt ist eine Abschlussarbeit (Prüfungsleistung) für den Ausbildungsberuf Fachinformatiker für Anwendungsentwicklung.

"Durch die Projektarbeit und deren Dokumentation soll der Prüfling belegen, dass er Arbeitsabläufe und Teilaufgaben zielorientiert unter Beachtung wirtschaftlicher, technischer, organisatorischer und zeitlicher Vorgaben selbstständig planen und umsetzen sowie Dokumentationen anfertigen, zusammenstellen und modifizieren kann." [Industrie und Handelskammer Infos](#)

Diese Dokumentation ist unter [Plantuml-With-Screenreader GitHub](#) veröffentlicht und zeigt Erweiterungen des Raumbuchungssystems.

## 1.1 Thema der Projektarbeit

Das Ziel dieses Projekts ist einen Abteilungsraumplaner als REST API und Minimales Frontend umsetzen, mit Hilfe von Middleware Samply.Beam (Abteilungsprojekt zur Kommunikation in strikten Kliniknetzwerkumgebungen <https://github.com/samply/beam>), um Meeting Room Booking System ( Meeting Room Booking System: <https://github.com/meeting-room-booking-system/mrbs-code>) zu erweitern und einfacher nutzbar zu machen.

## 1.2 Betriebsumfeld

Das Deutsche Krebsforschungszentrum (DKFZ) ist mit mehr als 3.000 Mitarbeiterinnen und Mitarbeitern in 90 Abteilungen die größte biomedizinische Forschungseinrichtung in Deutschland. Die Abteilung Federated Information Systems von Prof. Dr. Martin Lablans in der diese Projektarbeit durchgeführt wird, untersucht hierbei die Probleme die bei der Vernetzung von medizinischer Forschung auftreten. Hierbei entwickelt die Abteilung Software-Infrastruktur für Bsp. Datenschutz, Pseudonymisierung und interoperable Semantik zum Austausch von medizinischen Forschungsdaten.

In dem von der Abteilung Federated Information Systems entwickelten Software-Produkt Bridgehead ( <https://github.com/samply/bridgehead> ) wird Samply.Beam zum Austausch von Forschungsdaten und personenbezogenen Daten genutzt.

Der Zweck der Software [Bridgehead \(siehe Abbildung\)](#) ist es einen Zusammenschluss wissenschaftlicher Institute zu ermöglichen, um Forschungsdaten auszutauschen bzw. zu teilen und andere Software Komponenten einzubinden. Zentral wurde hier ein Netzwerk geschaffen in dem alle deutschen Biodatenbanken und internationale Partner vertreten sind, um pseudonymisiert Krebsdaten zu suchen und einzugrenzen. Diese Daten können dann bei unseren Partnern für Ihr Forschungsprojekt beantragt werden.

Die Software implementiert einen Deployment Mechanismus der flexibel Abteilungssoftware Komponenten und Skripte bereitstellt, für einen Bridgehead-Server.

Das Abteilungs-Software-Produkt Samply.Beam ( <https://github.com/samply/beam> ) ermöglicht Kommunikation in strikten Netzwerkumgebungen, es regelt die Kommunikation und nimmt so Komplexität von anderen Netzwerkkomponenten siehe [Beispiel Broker](#).

Im Rahmen dieser Arbeit wird ein Server zuerst einmal beim Broker registriert und ein Server\_Proxy übernimmt die Kommunikation mit dem Broker. Auf der Client Seite dient ein Client\_Proxy zur Kommunikation mit dem Broker. Der Broker stellt den Vermittler dar in diesem Netzwerk, so wird sichergestellt, dass nur registrierte Teilnehmer zugelassen sind.

Samply.Beam ist in Rust geschrieben, bietet Ende-Zu-Ende Verschlüsselung, Zertifikatsmanagement und lässt sich als REST API einbinden.

## 1.3 Projektbeschreibung

Die Abteilung (Federated Information Systems) hat einen Raumplaner für die Standorte Heidelberg und Mannheim, um flexibles Arbeiten an unterschiedlichen Arbeitsplätzen zu ermöglichen. Dieser Raumplaner ist ein GitHub Open Source Projekt namens Meeting Room Booking System: [mrbs-code](#). Dieses Raumplanungssystem wird schon über ein Jahr

genutzt und ist nur intern im DKFZ Netzwerk verfügbar. Nun soll dieser Dienst dahingehend überarbeitet werden, dass er auch außerhalb des VPNs genutzt werden kann. Die neue Software Lösung soll einfach bedienbar, alltagstauglich und leicht zugänglich sein und auf Samply.Beam als zentraler Infrastruktur aufbauen.

## 1.4 Projektziel

Bei der Umsetzung soll zentral Samply.Beam zum Einsatz kommen. Der Raumplaner soll zunächst in einem Testsystem verwendet werden. Statt einem Austausch von Patientendaten wie sie üblicherweise in den anderen Anwendungen der Abteilung ausgetauscht werden, sollen hier Raumbuchungsdaten ausgetauscht und dokumentiert werden.

Wenn die zu implementierende Anwendung später in einen produktiven Betrieb übergeht, könnte sie den Abteilungskollegen eine erhebliche Erleichterung in Ihrem Arbeitsalltag bringen und mit neuen Funktionalitäten eine erhebliche Zeiteinsparung.

## 1.5 Projektumfeld

Die Projektentwicklung wird von einem vollausgestatteten Mac M3 Pro Sonoma 14.4.1 PC-Arbeitsplatz mit VoiceOver (Screenreader) durchgeführt. Mit OpenSource Entwicklungsumgebung VSCode, Docker und Httpie.

Technologie und Bausteine sind Samply.Beam als Middleware und MRBSdas in diesem Projekt erweitert werden soll. Eingesetzt wird Docker als Lokale Entwicklungsumgebung, um Backend-Komponente als REST API und Frontend-Komponente Web Singlepage (html, JavaScript, css) zu entwickeln. Im Verlauf des Projektes wird GIT zur Versionsverwaltung verwendet und um die Software Lösung am Schluss bereitzustellen für die Abteilungskollegen. So sollen auch weiterführende Konzepte und Lösungen entwickelt werden zum Beispiel für eine mobile App.

*Tabelle 1: Projektumfeld*

| Rolle  | Name              |
|--|-------------------|
| Auftraggeber                                 | Dr. Tobias Kussel |
| Betreuer                                     | David Scholz      |
| Ansprechpartner zu Rust und Samply.Beam      | Jan Skiba         |
| Ansprechpartner Systemadministration         | Emil Simes        |
| Ansprechpartner Samply.Broker Administration | Torben Brenner    |

## 1.6 Projektschnittstellen

In der Projektlaufzeit von 80 Stunden werden folgende Schnittstellen hergestellt:

- Abteilungsraumplaner (Emil Simes)
- Abteilungsbroker (Torben Brenner)
- Implementierung von Raumplaner und Client über Samply.Beam (Dr. Tobias Kussel und Jan Skiba) [siehe Basis Samply.Beam](#).



## 2 Projektplanung

### 2.1 Erarbeitung der Anforderung

Im Gespräch mit Auftraggeber Dr. Tobias Kussel wurde festgelegt Samply.Beam zentral als Infrastruktur und Rust als Programmierungssprache zu nutzen. Es sollen auch konkret Konzepte entwickelt werden die Raumbelegungssoftware später auch noch als mobile App umzusetzen, im Idealfall als aufbauende Entwicklungsschritte. siehe [Projektauftrag](#)

### 2.2 Prüfung der Vorarbeiten

Im ersten Lehrjahr wurde von Denis Köther und mir der Abteilungsraumplaner aus einem GitHub Projekt [mrbs-code](#) bereitgestellt.

Außerdem wurde von Denis Köther und mir eine REST API in Java geschrieben, um ein digitales Türschild zu realisieren, dieses wurde jedoch nicht fertiggestellt.

Daniel Menzel (Student) hat in seinem Praktikum einen Prototyp in Rust geschrieben, dieser fokussierte sich auf Web Scraping des Abteilungsraumplaners.

Diese kommen für eine Weiterentwicklung in Frage, jedoch wurde in der Analyse geprüft ob ein neues Konzept die Anforderungen besser erfüllen wird.

### 2.3 Konzept Recherche

Zur Umsetzung sind folgende Programmiersprachen möglich:

Python, Java, Rust, PHP

| Kategorie                  | Gewichtung | Python Bewertung | Python Gewichtung | Java Bewertung | Java Gewichtung | Rust Bewertung | Rust Gewichtung | PHP-Bewertung | PHP-Gewichtung |
|----------------------------|------------|------------------|-------------------|----------------|-----------------|----------------|-----------------|---------------|----------------|
| Abteilungspräferenzen      | 40         | 1                | 41                | 3              | 43              | 3              | 43              | 1             | 41             |
| individuelles Know-how     | 10         | 2                | 12                | 2              | 12              | 3              | 13              | 2             | 12             |
| Bibliotheken               | 20         | 3                | 23                | 1              | 21              | 3              | 23              | 1             | 21             |
| Samply.Beam Kompatibilität | 30         | 1                | 31                | 1              | 31              | 3              | 33              | 1             | 31             |
| Gesamt Punkte              | 100        |                  | 107               |                | 107             |                | 112             |               | 105            |

Tabelle 2: Nutzwertanalyse Programmiersprache

Möglich wäre eine Entwicklung in Python so könnte diese leicht weiterentwickelt werden, durch nachfolgende Azubis. Weiterhin wäre eine cross-plattform Lösung inklusiv mobiler App ein innovativer Lösungsansatz.

Für Python würde sich [kivy](#) anbieten, diese Oberfläche ist jedoch nicht barrierefrei, daher wäre [BeeWare](#) eher geeignet.

In Rust würde sich [Tauri](#) als cross-plattform App anbieten mit mobiler App in der Alpha Version.

Die Umsetzung des Konzepts mithilfe dieser Frameworks wäre wünschenswert, weil diese einen großer Teil der optionalen Features unterstützen. Darauf muss aber aus Zeitgründen verzichtet werden. Beim gegebenen Zeitumfang bieten sich eher Flask in Python oder Axum in Rust, mit Docker als Basis an, da Docker auf jedem Desktop-Betriebssystem nutzbar ist und zur Entwicklung der Komponenten auch eingesetzt wird.

## 2.4 Personalplanung

Für das Projekt sind folgendes Personal mit zeitlichem Aufwand geplant:

| Name              | Tätigkeit  | Aufwand in Stunden |
|-------------------|--|--------------------|
| Martin Jurk       | Projektumsetzung   | 80                 |
| Dr. Tobias Kussel | Auftraggeber,<br>Projektanahme   | 5                  |
| Jan Skiba         | Ansprechpartner für<br>Samply.Beam, Code<br>Review, Samply.Beam<br>Deployment, Betreuung | 20                 |
| David Scholz      | Ansprechpartner,<br>Betreuung  | 5                  |
| Torben Brenner    | Zugriff auf<br>Abteilungsbroker-Server   | 1                  |
| Emil Simes        | Zugriff auf<br>Abteilungsraumplaner<br>Server, Administrative<br>Unterstützung           | 1                  |

Tabelle 3: Personalplanung

## 2.5 Sachmittelplanung

Für das Projekt sind folgende Sachmittel geplant:

| Sachmittel                   | Beschreibung  |
|------------------------------|---|
| Abteilungsbroker Test Server | zentraler Broker Test-Server für<br>Samply.Beam Konfigurationen und<br>neuen Komponenten mit öffentlicher<br>Domain |
| Büroplaner Server            | Abteilungsserver vom aktuellen<br>Büroplaner mit internen DKFZ-Domain   |
| Arbeitsplatz                 | Für Projekt Umsetzung (PC,<br>blindenspezifische Hilfsmittel I  |
|                              |   |

Tabelle 4: Sachmittelplanung Hardware

Für das Projekt sind folgende Sachmittel an Software geplant:

| Sachmittel Software | Beschreibung  |
|---------------------|---|
| Samply.Beam         | Abteilungsprojekt                                   |
| Visual Studio Code  | IDE   |
| Httpie              | API-Test  |
| Docker              | Containersoftware für die Umsetzung<br>des Projekts |

## 2.6 Kostenplanung

Die aufgelistete Hardware ist bereits als bestehendes System zu betrachten, daher wird keine Kostenrechnung seitens der Hardware beachtet.

Die Software wird mit Ausnahme von MacBook Pro, Office Paket und blindenspezifischen Hilfsmitteln werden mit in die Ausstattung mit einberechnet.

Ein Azubi im 3. Lehrjahr wird laut TV-L BBiG (Stand: 2024) mit 1.190,61 € Bruttogehalt im Monat bezahlt. Dabei arbeitet der Azubi nur ungefähr 150 Tage im Jahr im Betrieb.

Für den Mitarbeiter in der Entwicklung mit Dokortitel, wird in TV-L 2024 mit E13 gerechnet. Dies bedeutet ein Monatsbruttogehalt von 4.188,38 € in der 1. Stufe und bei einem IT-Mitarbeiter mit Fachinformatiker Ausbildung in TV-L 2024 mit E8 2.946,46 €.

Hier werden ebenso Lohnnebenkosten von 21% gerechnet. Abzüglich Urlaubsanspruchs von 30 Tagen und Wochenenden werden hier 220 Tage als effektive Arbeitszeit gerechnet. Für Büromaterialien und Ausstattung wird ein Kostensatz von 20€/h gewählt. Darunter zählen Arbeitsplatz mit PC, Lizenzen und weitere Ausstattungen.

$$\frac{(\text{Bruttojahresgehalt} + \text{jährliche Lohnnebenkosten})}{(\text{Arbeitstage} * \text{Arbeitsstunden pro Tag})} + \text{Ausstattung pro Stunde} = \text{Stundensatz}$$

## 2.7 Amortisationsrechnung

Wir nehmen an, dass für die neue Raumbelegungssoftware über Samply.Beam eine zentrale Organisation von 7 Stunden im Jahr entfallen pro Mitarbeiter. So setzen wir eine Einsparung von 2 Minuten am Tag von jedem Mitarbeiter, durch entfallen des VPN-Zugriff und neuen Features der Raumbelegungssoftware.

Für einen wissenschaftlichen Mitarbeiter wäre diese

$$4774,66 \text{ €} / (7 \times 54,55 \text{ €}) = 4774,66 \text{ €} / 381,85 \text{ €} = \text{ca. 12,5 Jahre Amortisationsdauer}$$

Für die Abteilung Federated Information Systems aktuell:

| Personal / Bezeichnung           |                                |
|----------------------------------|--------------------------------|
| 24 wissenschaftliche Mitarbeiter | Einsparung 43,64 € pro Tag     |
| Gesamt Kosten                    | 4774,66 € Kosten Projekt       |
| Amortisationsdauer               | 109,41 Tage == ca. halbes Jahr |

Tabelle 5: Amortisationsrechnung

So lohnt sich die Umsetzung der neuen Raumbelegungssoftware über Samply.Beam nach einem halben Jahr.

## 2.8 Projektphasen- und Zeitplanung

In diesem Fall hat sich das Wasserfallmodell angeboten da es klare Rahmenbedingungen gibt. Dieses lineare sequentielle Modell zur Softwareentwicklung, kann mit Meilensteinen gut genutzt werden den Auftraggeber so in das Projekt einzubinden. Diese Phasen bieten sich an:

Analyse, Entwurf, Implementierung, Test, Abnahme

| Projektphasen          | Zeit |
|------------------------|------|
| Projektinitialisierung | 1    |
| Planung                | 3    |
| Analyse                | 8    |
| Entwurf                | 7    |
| Implementierung        | 37   |

|               |    |
|---------------|----|
| Test          | 3  |
| Abnahme       | 2  |
| Dokumentation | 15 |
| Puffer        | 4  |
| Insgesamt     | 80 |

Tabelle 6: Grober Zeitplan und Meilensteine

## 2.9 Abgleich der Anforderung mit Auftraggeber

Der Auftraggeber Dr. Tobias Kussel erwartet eine Evaluation und Analyse zu folgenden Themen mit konkreten Konzepten.

### 2.9.1 Evaluation und Analyse Themen

- Umsetzung in Rust
- Authentication der Nutzer klären
- Definition REST API
- Frontend minimal Entwurf
- Integration Beam-Proxy und Frontend
- Prüfung der Datenbank des Abteilungsraumplaners
- Konzepte für eine App Lösung entwickeln
- ermitteln möglicher Bibliotheken für eine mobile App oder minimal Lösung

## 3 Analysephase

Nach der Projektplanung kann nun ermittelt werden welche Vorarbeiten im Projekt miteinfließen können. So müssen auch verschiedene Konzepte betrachtet werden die sich zur Umsetzung am besten eignen. Zentrale Prüfungskriterien ist der zeitliche Umfang und die Einbindung von Samply.Beam um dem Projektziel gerecht zu werden. Ebenso wie ein guter Zugriff und Nutzbarkeit für alle Betriebssysteme.

### 3.1 Ist-Analyse

Aktuell ist MRBS [mrbs-code](#) für die Abteilung eingerichtet. Diese ist in PHP entwickelt und nur intern im Netzwerk verfügbar. Dazu gibt es Vorarbeiten, aber diese sind Prototypen.

Die Datenbank (MYSQL) ist wie folgt strukturiert:

#### [Mrbs Datenbank](#)

#### 3.1.1 Standorte und Räume in der Datenbank

Aktuell gibt es einen Standort in Heidelberg und Mannheim.

#### 3.1.2 Datenbank Registrierung für Arbeitsplatz (Büro)

Zur Reservierung eines Raums muss in "mrbs\_participants" mit der entsprechenden entry\_id eine Buchung eingetragen werden.

Beispiel Einträge:

```
+---+-----+-----+-----+-----+
| id | entry_id | username | create_by | registered |
```

```
+----+-----+-----+-----+-----+
| 5 | 258 | test | test | 1652945566 |
| 7 | 745 | test | test | 1652961919 |
| 8 | 747 | test | test | 1652962479 |
| 9 | 1137 | app1 | app1 | 1652962853 |
| 10 | 975 | mj | mj | 1652963685 |
| 11 | 747 | app2 | app2 | 1653032406 |
| 15 | 1150 | cc | cc | 1653308419 |
| 16 | 1002 | app1 | app1 | 1656328038 |
| 17 | 1155 | martin | martin | 1713513920 |
| 18 | 1198 | martin | martin | 1713520446 |
| 19 | 1155 | app1 | app1 | 1713520470 |
| 22 | 1533 | martin | martin | 1713881097 |
| 25 | 1550 | martin | martin | 1715340694 |
| 26 | 1176 | martin | martin | 1715340724 |
| 28 | 1222 | a..1 | a..1 | 1715587968 |
| 29 | 1222 | app1 | app1 | 1715597477 |
| 30 | 1223 | app1 | app1 | 1715687521 |
+----+-----+-----+-----+-----+
```

17 rows in set (0,01 sec)

Die ID für entry\_id muss in "mrbs\_entry" ermittelt werden um diese in "mrbs\_participants" einzutragen, siehe Ausschnitt aus der Terminliste. Diese werden für jeden Raum immer von 8 bis 16 Uhr vorbelegt, als automatische Tagesbelegung vom Administrator.

```
+----+-----+-----+-----+
| id | name | room_id | create_by |
+----+-----+-----+-----+
| 258 | Büro | 13 | dk |
+----+-----+-----+-----+
```

1 row in set (0,00 sec)

So gestaltet sich der Ablauf einen Arbeitsplatz zu buchen, über SQL-Statements direkt mit dem Datenbankmanagementsystem. Es müssen nur folgende Felder eingetragen werden:

entry\_id | username | create\_by

```
insert into mrbs_participants(entry_id, username, create_by)
Values(258, "erwin.müller", "erwin.müller" );
```

## 3.2 Konzepte

### 3.2.1 Variante A dezentrales Netzwerk (kleine Brückenköpfe)

Die Idee bei dieser Variante ist zentral Docker zu nutzen als Plattform für das Produktivsystem. So können alle Nutzer, unabhängig vom Betriebssystem die Client-Komponenten nutzen. Oder diese im Heimnetz hosten auf einem Raspberry rum Beispiel.

Die Initialisierung des Samply.Proxy am Samply.Broker wird mit Hilfe des Zertifikats umgesetzt. So wird eine Authentifizierung gewährleisten.

Siehe folgende Abbildung [dezentrales Netzwerk](#).

Hier gibt es den "kleinen Brückenkopf" als Client der vom Nutzer installiert wird und dieser kann dann zentral über den Broker mit der Backend-Komponente kommunizieren.

### 3.2.2 Variante B mit mobiler App:

In Variante B soll zentral für den Nutzer der einfache Zugriff über eine Handy-App umgesetzt werden. Beispiel [mobile App](#).

Hier wäre nur eine Kommunikation mit dem zentralen Service mit zentralem Proxy möglich. Dort müsste eine Authentifizierung mit Bsp. Keycloak umgesetzt werden. So könnte eine mobile App auch beispielsweise in Python (Framework Kivy oder Beeware) entwickelt werden.

### 3.2.3 Tauri

Tauri ist ein App-Build-Toolkit, mit dem Software für alle wichtigen Desktop-Oberflächen umgesetzt werden können. Dies basiert auf Rust und nutzt als Frontend JavaScript, JavaScript Frameworks oder Svelte. Eine Umsetzung als mobile App ist auch Möglich und aktuell im Alpha Release.

Beispiel [Tauri mobile App](#) Umsetzung.

Diese Umsetzung hat den Vorteil, dass der Samply.Proxy direkt in Rust eingebunden werden kann.

### 3.2.4 Variante C zentrale Client-Komponente:

Variante C sieht vor die Client-Komponente zentral zu hosten mit Beam.Proxy und Beam.Broker. Der Nutzer loggt sich über den Browser ein, wobei die Authentifizierung jedoch auf andere Weise zu Implementieren wäre als in den Varianten A und B. . Beispiel [zentrale Client Komponente](#).

## 3.3 Soll-Analyse

Die Zentralen Gesichtspunkte und Anforderung für das Projekt sind Benutzerfreundlichkeit, Zeitumfang für die Implementierung, der Benutzerkreis und benötigte Funktionen der Raumbelegung. Die Benutzerfreundlichkeit enthält einen Zugriff ohne VPN, sowie eine schnelle und klare Übersicht welche Kollegen sich für heute am Standort befinden oder an einem bestimmten Tag. Der Benutzerkreis soll aktuell nur die Abteilungskollegen umfassen und MRBS das aktuelle Raumbuchungssystem erweitern. Das Endprodukt soll in Rust umgesetzt werden für die bessere Anbindung an Samply.Beam und da es in der Abteilung eine wichtige Schlüsseltechnologie darstellt.

Mit dem Auftraggeber Dr. Kussel wurde folgende Zeitplanung abgestimmt. siehe [Projekt Zeitplanung](#)

Die Funktionen der Komponenten sollen so umgesetzt werden, dass diese noch später für eine Umsetzung mit Tauri mobile App nutzbar sind.

## 4 Entwurfsphase

### 4.1 Auswahl der Bibliotheken

Die Komponenten sollen in Rust umgesetzt werden. Für den Frontend Client bietet sich an [Yew](#) oder [Axum](#) mit JavaScript oder [Svelte](#). Yew ist ein Rust Web Framework dieses kann direkt mit Macros die Html Seite rendern und ausliefern. Für Axum mit Svelte gibt es ein [Template](#), dabei stellt Axum die Server-Side Funktionen und Svelte das Frontend. Svelte wird kompiliert in reines JavaScript und hat viele Macros zur Vereinfachung von Web Komponenten. Aber es wäre auch Möglich selbst JavaScript direkt umzusetzen.

Siehe [Nutzwertanalyse zu den Bibliotheken](#).

Für das Backend gibt es klare Abteilungspräferenzen für Axum und grundsätzliche für alle Bibliotheken von den [Tokio](#) Entwicklern. So fällt die Wahl einer ORM-Bibliothek(Object Related Mapper) für den Datenbankzugriff auf [SeaORM](#), da diese auch von den Tokio Entwicklern stammt.

Wegen dem Zeitumfang von 37 Stunden zur Implementierung der Software wird die minimale Lösung mit Axum und Svelte umgesetzt.

### 4.2 Ermittlung des Funktionsumfang

Es werden folgende Routen oder Funktionen benötigt:

GET /overviewday mit Parameter "day" als Datum String. Dieser soll eine Übersicht von Standorten, Räumen und Teilnehmer zurückgeben. Wie Siehe [Raumbelegung](#).

GET /room/id hier ist "id" die Raum ID. Gibt Raum mit Teilnehmern zurück.

POST /joinroom/id hier ist ID die Entry ID. So wird der Raum mit vorbelegtem Eintrag gebucht.

POST /joinrooms/id mit Parameter ID als Raum ID "days" als Array aus Datum Strings. Dies ist die Erweiterung und soll die Möglichkeit bieten Buchungen für mehrere Termine durch zu führen.

DELETE /deletejoin/id mit dem Parameter "id" als Entry ID . Um die Teilnahme zu löschen.

### 4.3 Deployment

Samply.Beam stellt eine Demo zu Verfügung um die Entwicklung von Komponenten in Docker-Containern zu vereinfachen. Diese wird zur Entwicklung der Backend- und Client-Komponenten genutzt und angepasst. Außerdem wird MRBS als Docker Container lokal mit importierter Datenbank als SQL-Backup aufgesetzt und eingespielt.

So kann erstmal das Backend entwickelt werden mit API-Client HTTPIE (API-Testing-Tool). Um die Abfragen auf die Datenbank entwickelt, ohne Samply.Beam.

Im zweiten Schritt wird Samply.Beam als Vermittler eingebunden. Siehe [minimal Umsetzung](#)

### 4.4 Entwurf Front - und Backend

Die Backend-Connector Komponente muss folgende Funktionen umsetzen. siehe folgende [Routen in Modul](#).

```
get_overview_day(db, customday: Option<String>)
postjoinroom(db, entry_id: i32, user_name)
post_joinrooms(db, room_id: i32, days: Vec<String>, user_name)
delete_participant(db, entry_id: i32)
```

Weiterhin muss die Komponente für die gewählte Tagesansicht eine Datenstruktur implementieren.

Siehe [Datenstruktur Overview](#)

Die den Standort mit Räumen und die eingetragenen Kollegen umsetzt. Dort muss die Standorte(mrbs\_area), Räume(mrbs\_room) und Teilnehmer(mrbs\_participants) mit den jeweiligen IDs enthalten sein. Zusätzlich muss noch in den Räumen "entry\_id" die ID der vorbelegten Zeiten und das Buchungslimit mit enthalten sein. Diese wird zum buchen eines Raumes benötigt, die Zeitfenster werden in MRBS schon vorgelegt von 8 bis 16 Uhr da in der Abteilung nur tagesweise ein Arbeitsplatz reserviert wird.

Für den Client wird das [Axum Template](#) mit Svelte genutzt diese muss nur angepasst werden. Die Client-Frontend Komponente kann dynamisch mit Svelte die Datenstruktur des Backend-Connector als JSON nutzen um eine minimale Übersicht zu gestalten.

Siehe [Entwurf Übersicht Darstellung](#).

Hier werden die Standorte verschachtelt mit den jeweiligen Räumen und deren Teilnehmer angezeigt als dynamische Liste.

## 4.5 Schnittstellen zu Samply.Beam

Für die Anbindung an Samply.Beam muss die Client-Backend Komponente einen Task an den Samply.Broker senden, in der [Samply.Beam Bibliothek](#) werden dafür Funktionen bereitgestellt. Der [Samply.Beam Task](#) ist ein "struct", das ID, TO, FROM, BODY und weitere Felder enthält.

So gibt es auch für den Backend-Connector dort Funktionen, die auf Tasks hören und einen [Samply.Beam Result Task](#) für eine Rückantwort an die Client-Backend Komponente. siehe [Task erstellen](#) und [Task beantworten](#)

# 5 Implementierung

## 5.1 Lokale Testumgebung

Zur Entwicklung wurde lokal MRBS als Docker Container eingerichtet. Siehe [Docker-Compose-Datei](#).

## 5.2 Datenbankanbindung

Durch die Nutzung des Frameworks SeaORM muss die vorhandene Datenbank erst durch den SeaORM Client eingelesen werden.

```
sea-orm-cli generate entity -u mysql://user:password@localhost:3306/mrbs -o entities/
```

So werden durch SeaORM für jede Tabelle eine entsprechende Datenstruktur und Beziehung generiert.

Für die Tagesansicht muss die Datenstruktur "EntryRelatedData" mit den entsprechenden Daten befüllt werden. Dafür werden erst alle Standorte in "mrbs\_area" ermittelt und in "mrbs\_room" alle verbundenen Räume für die Grundstruktur. Ebenso alle Teilnehmer in "mrbs\_participants", dies wurde wegen der Komplexität nicht als JOIN umgesetzt. siehe [Tagesansicht Code](#)

Alle Datenstrukturen wurden in eine eigene Bibliothek gepackt "[shared\\_types\\_planner](#)", da diese in beiden Komponenten benötigt werden.

Für den Umgang mit dem Datum wurde ein String mit dem Format nach [rfc3339](#)

"2024-05-20T08:00:00Z" gewählt, da dieser einheitlicher genutzt werden kann und besser lesbarer ist als Unix Time Stamps.



Für die Funktion zum Registrieren und Löschen der Buchung des Arbeitsplatzes in entsprechendem Raum. siehe [join](#) und [delete](#)

Zum Testen der Komponente wurde eine simple REST API über Axum erstellt und mit dem HTTPIE Client getestet. Für das endgültige Projekt wurden nur die Funktionen übernommen, da dies zu Zeitaufwändig und Komplex mit der Einbindung in Samply.Beam wurde. siehe UML zu

[BackendConnector](#)

## 5.3 Client Front- und Backend Komponente

Mit dem simplen und flexiblen Notationssystem von Svelte Konnte die minimale Tagesansicht direkt mit Schleifen umgesetzt werden. siehe [Tagesansicht Code](#)

Die REST API Anfragen wurden in JavaScript umgesetzt, diese enthält Funktionen zum Buchen und Löschen. siehe [JavaScript Anfragen](#)

Das Backend des Clients nutzt die Vorlage des Axum Templates und implementiert mit API-Token die Kommunikation zwischen Client Backend und Client Frontend. Siehe [Routen](#) und [BeamRequestTask](#).

## 5.4 Samply.Beam Anbindung

### 5.4.1 Client Samply.Beam

Um die benötigten Parameter im "body" Feld des Task über Samply.Beam zu senden, wurde in der eignen Bibliothek "shared\_types\_planner" eine entsprechende Datenstruktur mit diesen Feldern angelegt. siehe UML [SharedTypes](#)

Wegen Problemen beim Error Handling wurden dort auch "ErrorTypes" und "ResponseTypes" als Enum angelegt, diese können so einfach als "Result<ResponseType, ErrorType>" Ergebnis der Anfrage zurückgegeben werden. Als Basic dient Module std:result

```
enum Result<T, E> {  
    Ok(T),  
    Err(E),  
}  
  
Result<ResponseType, ErrorType>
```

Um einen Task in Samply.Beam zu erstellen muss folgende Funktion eingebunden werden. Die Datenstruktur zu [Beam Tasks ist hier](#) zu sehen. Dieser benötigt den Verfasser und Empfänger aus der Konfiguration. Für die "task\_id" wird eine UUID v4 (Universally Unique Identifier) erzeugt zur Identifizierung dieses Task im späteren Verlauf.

```
let task = create_beam_task(task_id, param);  
  
BEAM_CLIENT  
    .post_task(&task)  
    .await  
    .map_err(|_error| Json(ErrorType::BeamNoTaskCouldBeCreated));
```

So muss auch der Task mit der entsprechenden ID mit dieser Funktion angefragt werden, ob ein [Task Ergebnis](#) schon verfügbar ist.

```
let res = handle_listen_to_beam_tasks::<Result<ResponseType, ErrorType>>(task_id)
```

```
.await  
.map_err(|_error| Json(ErrorType::NoBeamResponse))?  
.body;
```

#### 5.4.2 BackendConnector Samply.Beam

Die Backend Komponente nutzt nun auch Funktionen der Samply.Beam Bibliothek die auf einkommende Tasks vom Beam.Broker hören. Diese erwartet den Typ des "body" Felds hier "PlannerParams". Nun kann die Anfrage beantwortet werden.

```
let task = BEAM_CLIENT  
.poll_pending_tasks::<PlannerParams>(&BlockingOptions::from_count(1))  
.await  
.ok()  
.and_then(|mut r| r.pop());
```

Nun werden die Funktionen der Datenbankabfrage des Backend genutzt. Dies wird typisch für Rust mit einem match-Statement auf das Enum "Endpoints" sichergestellt. Mit dem Ergebnis wird ein [Result Task](#) erstellt.

```
let taskresult = TaskResult {  
    from,  
    to,  
    task: id,  
    status: beam_lib::WorkStatus::Succeeded,  
    body: taskresponse,  
    metadata: serde_json::Value::Null,  
};
```

Unter der erhaltenen UUID wird ein Result Task zurückgesendet.

```
BEAM_CLIENT  
.put_result(&taskresult, &for_task_id)  
.await  
.unwrap();
```

## 6 Projektabschluss

### 6.1 Soll-Ist-Vergleich

Die geplanten Ressourcen und Kosten konnten eingehalten werden. Durch die Vorbereitungsphase für die schriftliche Abschlussprüfung und eine Einführung in Rust Error Handling durch meinen Ausbilder hat sich der Abgabetermin verzögert. siehe Zeitablauf als Gantt-Diagramm

Die Pufferzeit wurde komplett in der Implementierungsphase genutzt, sonst gab es keine Abweichungen.

## 6.2 Übergabe

Die Umsetzung der Raumbelegungssoftware hat alle Anforderungen erfüllt. Die Anbindung an Samply.Beam und Installation war erfolgreich. Ein zusätzliches Feature wie das Buchen von den Arbeitsplätzen für die nächsten Wochen an entsprechenden Wochentagen wird noch später umgesetzt. Für eine Umsetzung als mobile App in Tauri wurde eine eigene Bibliothek erstellt, die eine Weiterentwicklung ermöglichen. so wurden auch die Optionalen Ziele des Auftraggeber Dr. Tobias Kussel umgesetzt.

Abteilungsintern wurde ein GitLab Repository erstellt eingerichtet mit Kundendokumentation in Englisch mit Installationsanweisung und Nutzungshinweise.

## 6.3 Ausblick und Fazit

Die entwickelte Raumbelegungssoftware bietet nun eine gute Übersicht und einen schnellen Zugriff bei der Arbeitsplatz Planung. Die Open Source Software Samply.Beam hat sich als gute Basis erwiesen und kann für vielfältige Anwendungen genutzt werden. Für mich war es eine angenehme und spannende Herausforderung Samply.Beam einzusetzen.

Die Frontend -, Backend - und Kommunikationskomponenten stellen ein gutes und umfassendes Thema in allen Bereichen der Softwareentwicklung, Förderierung, Sicherheit und Netzwerkcommunication dar. Der erfolgreiche Abschluss kann nun für weitere Entwicklungen und Features, wie mobile App genutzt werden. Die Programmiersprache Rust hält viele spannende Herausforderungen für einen Auszubildenden bereit und legt eine gute Basis als Backend-Entwickler.

## Literaturverzeichnis

Docker <https://www.docker.com>

Rust <https://doc.rust-lang.org/std/index.html>

Samply.Beam <https://github.com/samply/beam>

Rust Cookbook <https://rust-lang-nursery.github.io/rust-cookbook/>

Rust SeaORM Doku <https://www.sea-ql.org/SeaORM/>

Svelte <https://svelte.dev>

Rust Tokio <https://tokio.rs>

Rust Axum <https://github.com/tokio-rs/axum>

Rust Axum Svelte Template <https://github.com/jbertovic/svelte-axum-project>

Httpie <https://httpie.io>

MRBS <https://github.com/meeting-room-booking-system/mrbs-code>

## 7 Anlagen

## 7.1 Broker Beispiel

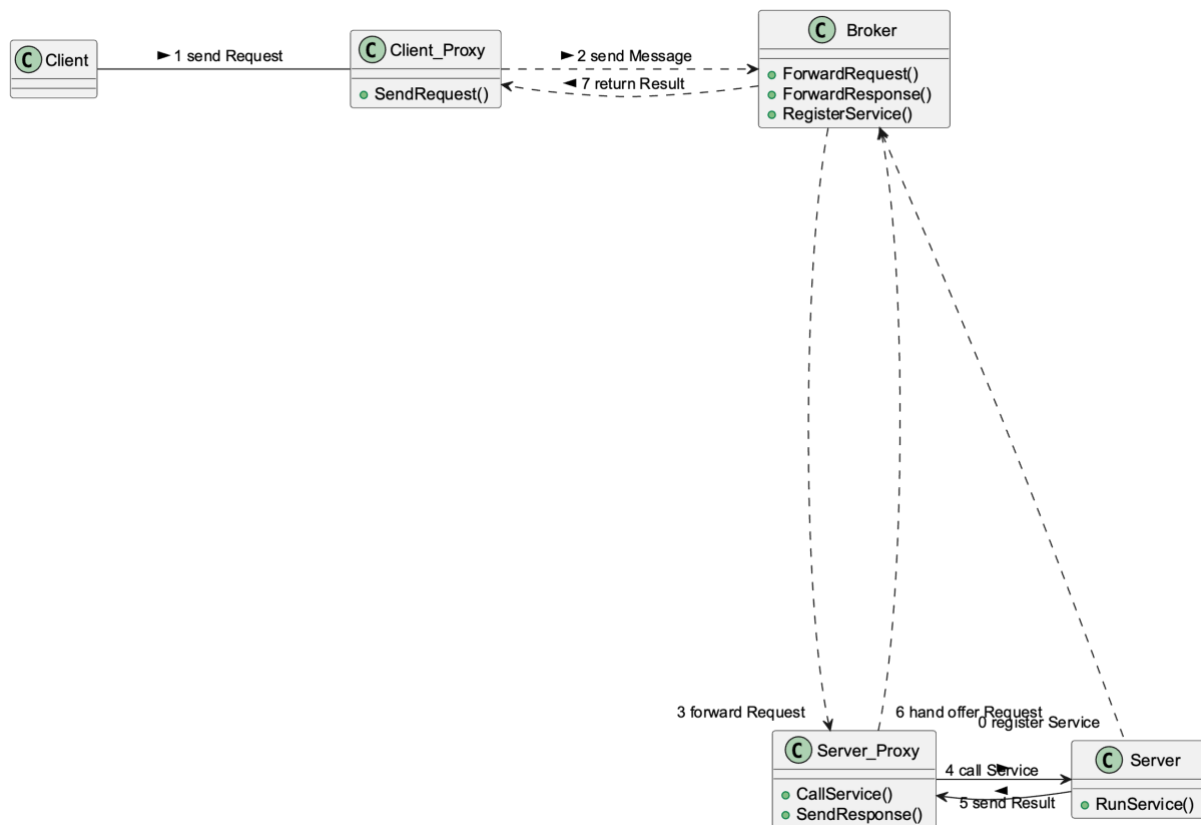


Abbildung 1: Broker Beispiel

## 7.2 Bridgehead

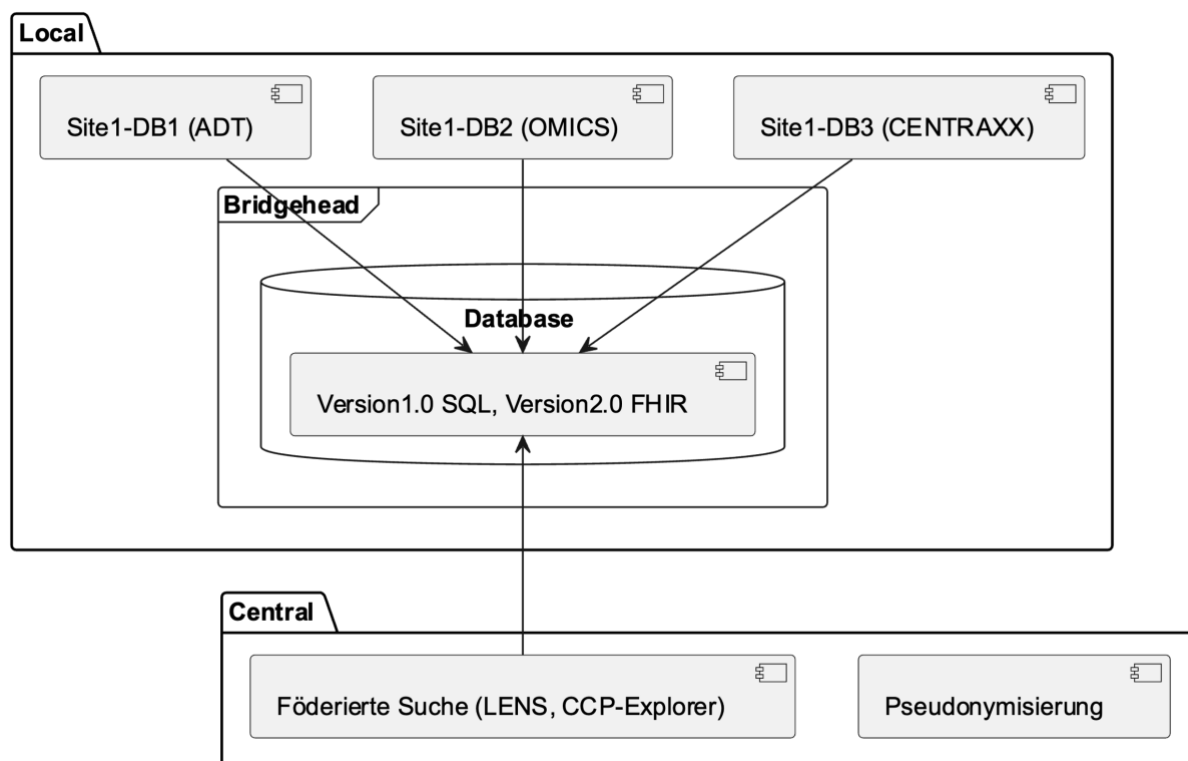


Abbildung 2: Bridgehead

### 7.3 Basis Beam Beispiel

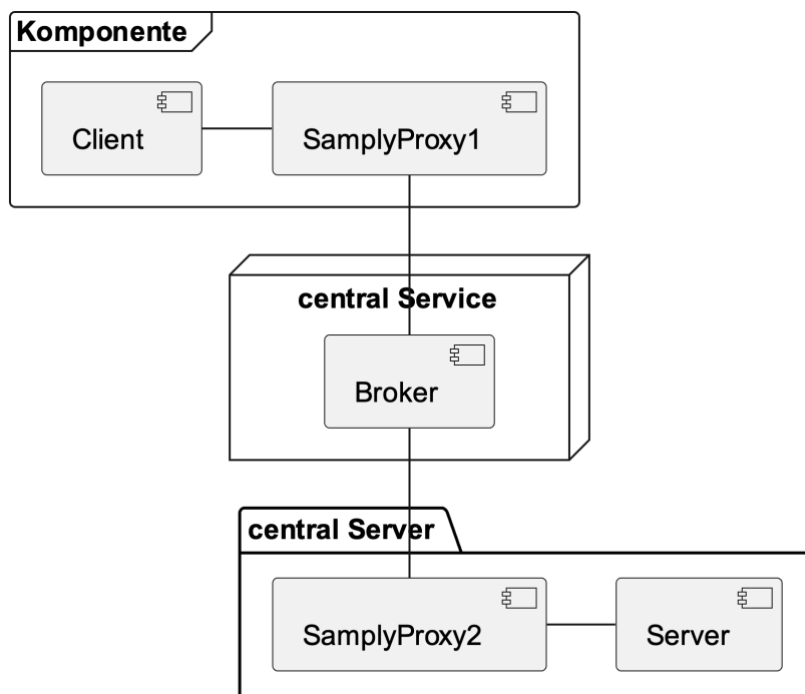


Abbildung 3: Basis Beam Beispiel

## 7.4 Mrbs Datenbank

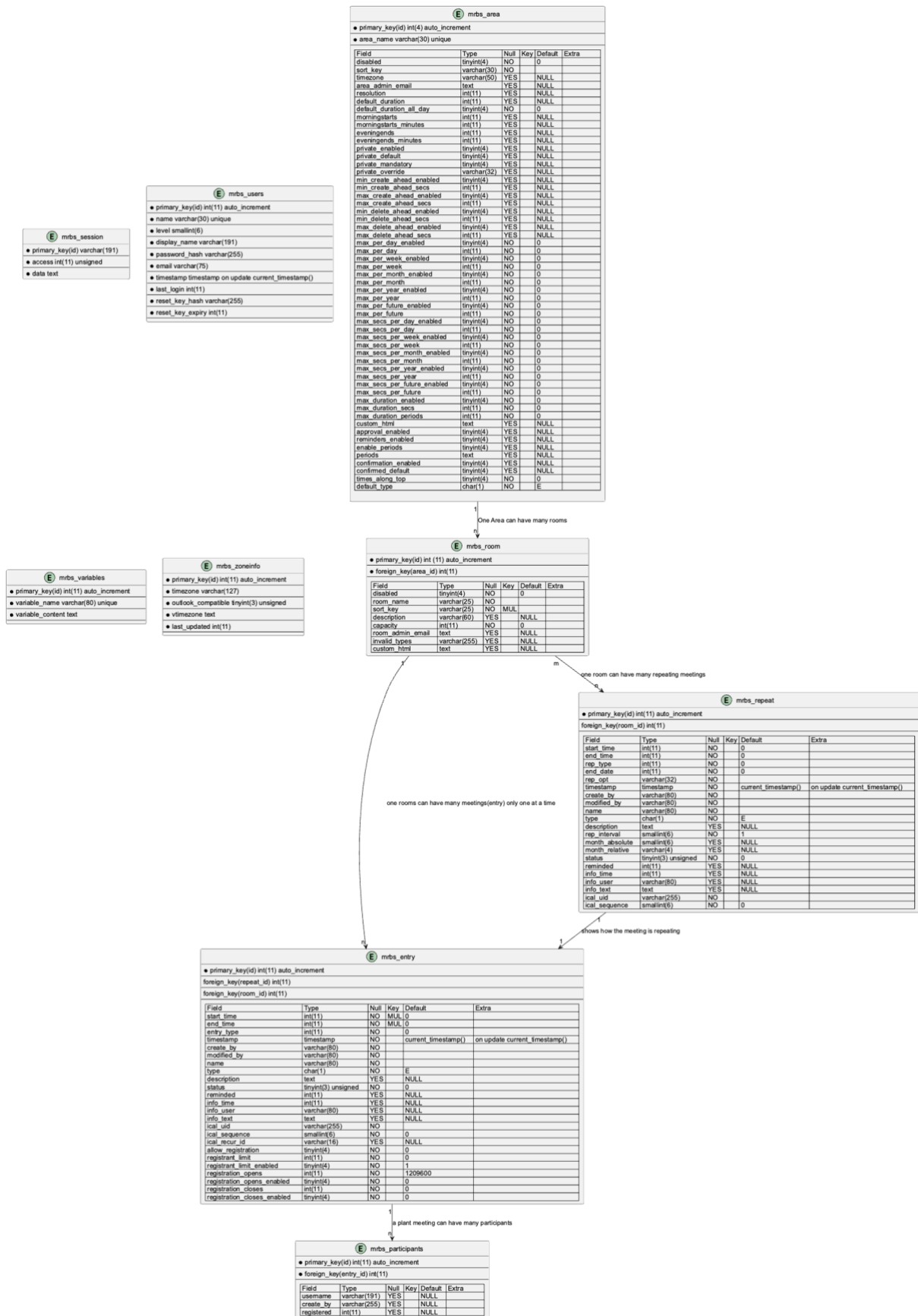


Abbildung 4: Mrbs Datenbank

## 7.5 dezentral Netzwerk kleine Brückenköpfe

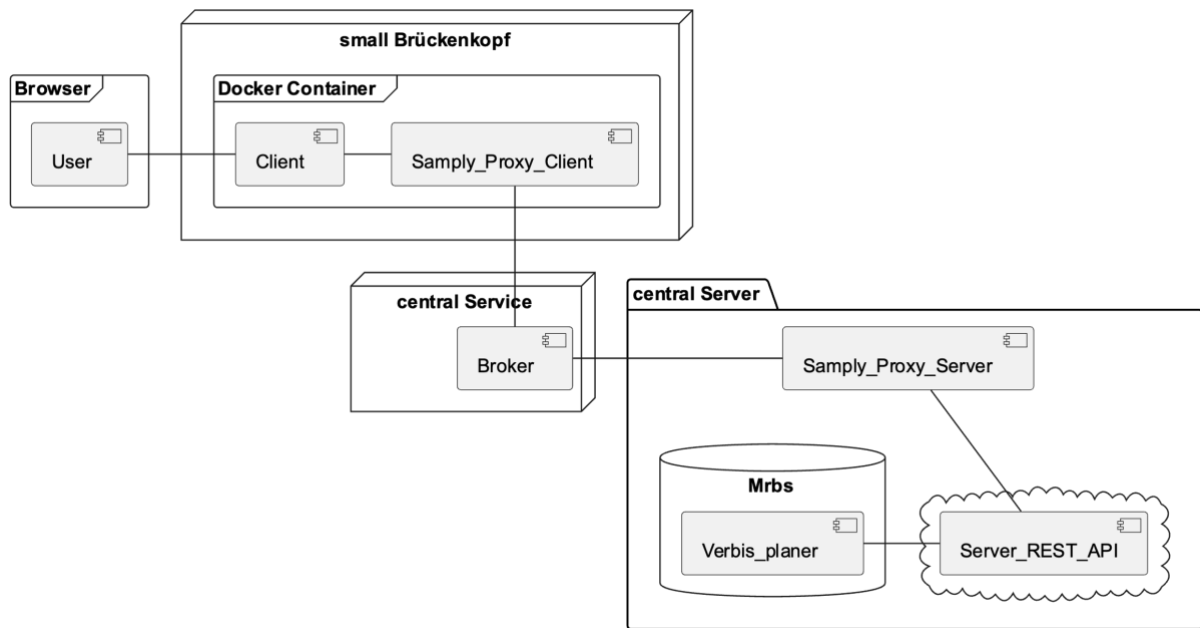


Abbildung 5: kleine Brückenköpfe

## 7.6 Mobile App

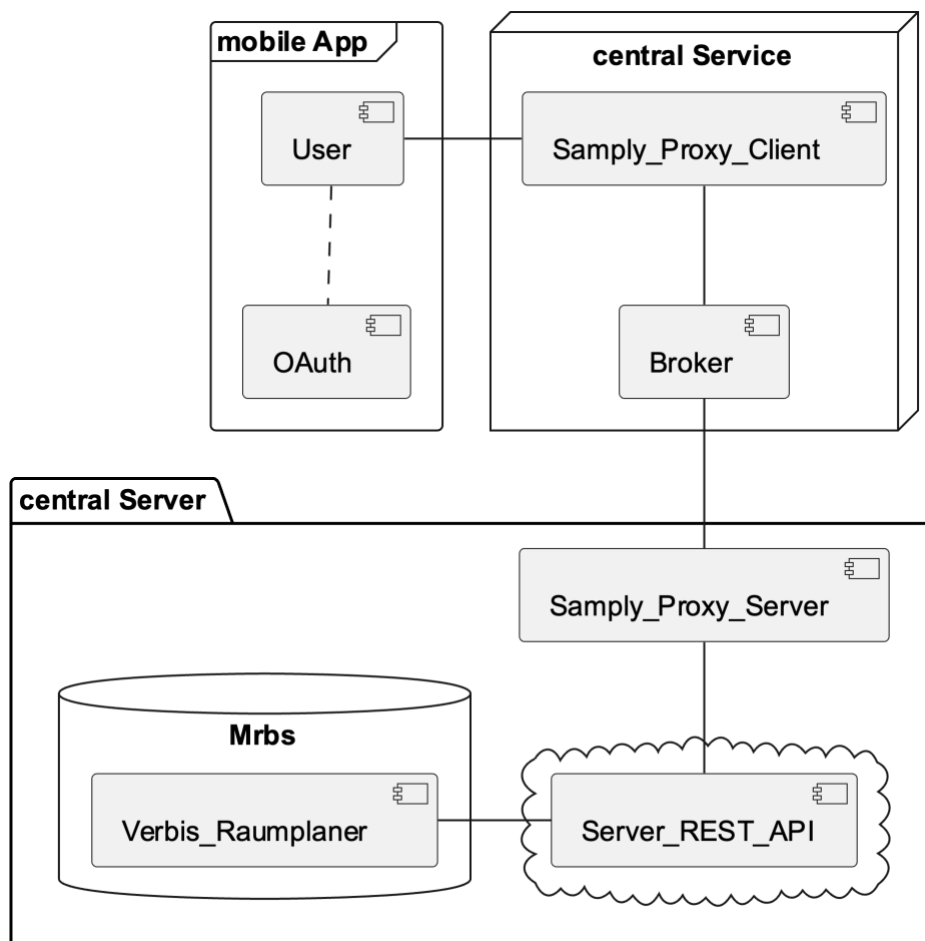


Abbildung 6: mobile App Basis

## 7.7 Mobile App Tauri

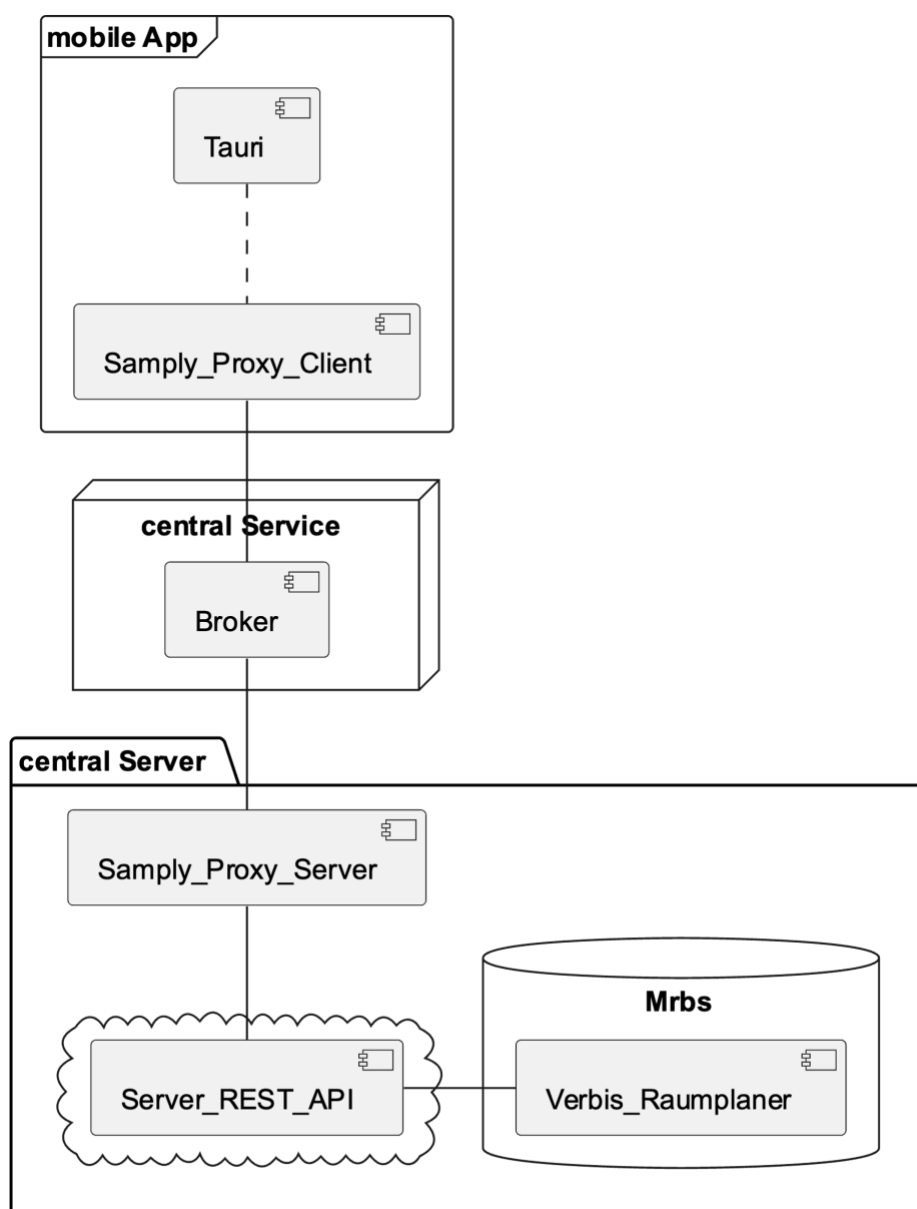


Abbildung 7: Tauri App



## 7.8 zentrale Client Komponente

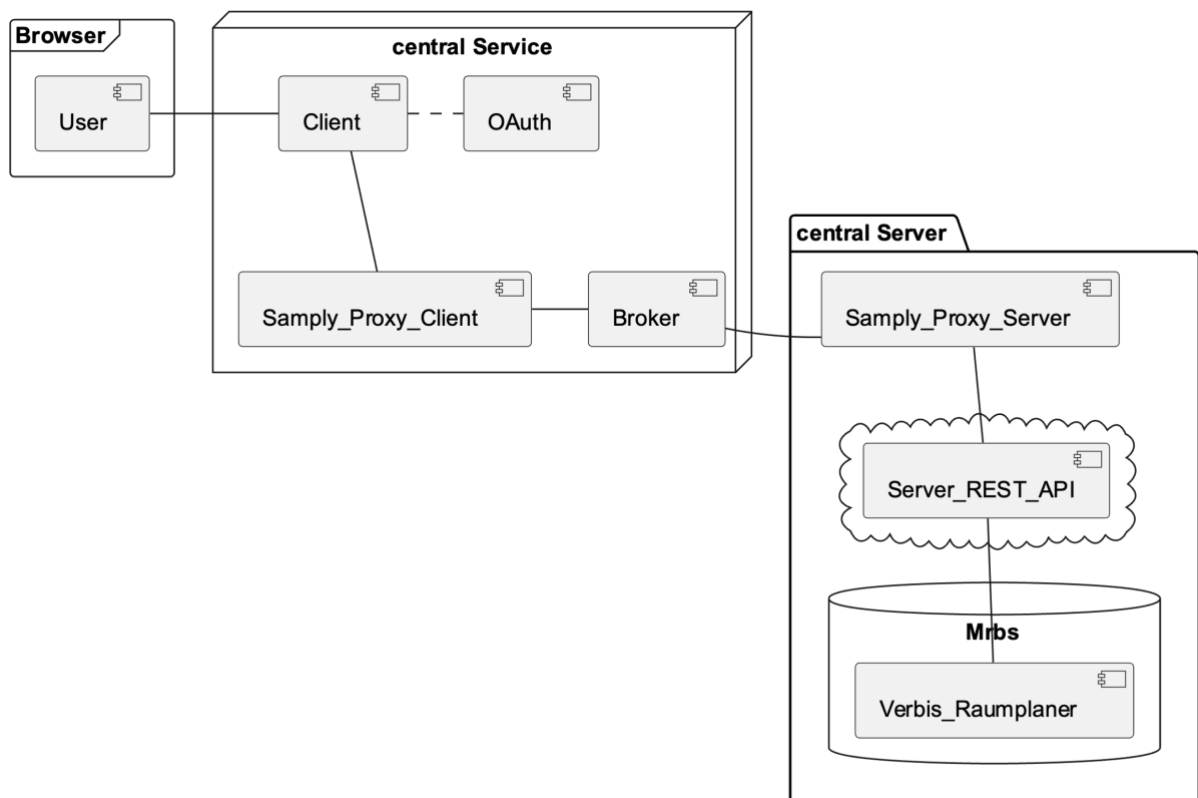


Abbildung 8: zentrale Client Komponente

## 7.9 Nutzwert Analyse Frontend Client

| Kategorie                        | Pkt | Yew<br>Bewert<br>ung | Yew<br>Gewicht<br>ung | Tauri<br>Bewert<br>ung | Tauri<br>Gewicht<br>ung | Axum mit<br>Svelte<br>Bewertung | Axum mit<br>Svelte<br>Gewichtung | Axum mit<br>JavaScript<br>Bewertung | Axum mit<br>JavaScript<br>Gewichtung |
|----------------------------------|-----|----------------------|-----------------------|------------------------|-------------------------|---------------------------------|----------------------------------|-------------------------------------|--------------------------------------|
| Abt. Pref.                       | 10  | 1                    | 11                    | 3                      | 13                      | 3                               | 13                               | 1                                   | 11                                   |
| Eigene<br>Kenntnisse             | 20  | 1                    | 21                    | 2                      | 22                      | 2                               | 22                               | 3                                   | 23                                   |
| Samply.Beam<br>kompatible        | 20  | 2                    | 22                    | 3                      | 23                      | 2                               | 22                               | 2                                   | 22                                   |
| Zeitungsfang<br>für<br>Umsetzung | 50  | 1                    | 51                    | 1                      | 51                      | 3                               | 53                               | 1                                   | 51                                   |
| Gesamt<br>Punkte                 | 100 |                      | 105                   |                        | 109                     |                                 | 110                              |                                     | 107                                  |

Tabelle 7: Nutzwertanalyse Client

## 7.10 Zeitablauf des Projekts als Meilensteine

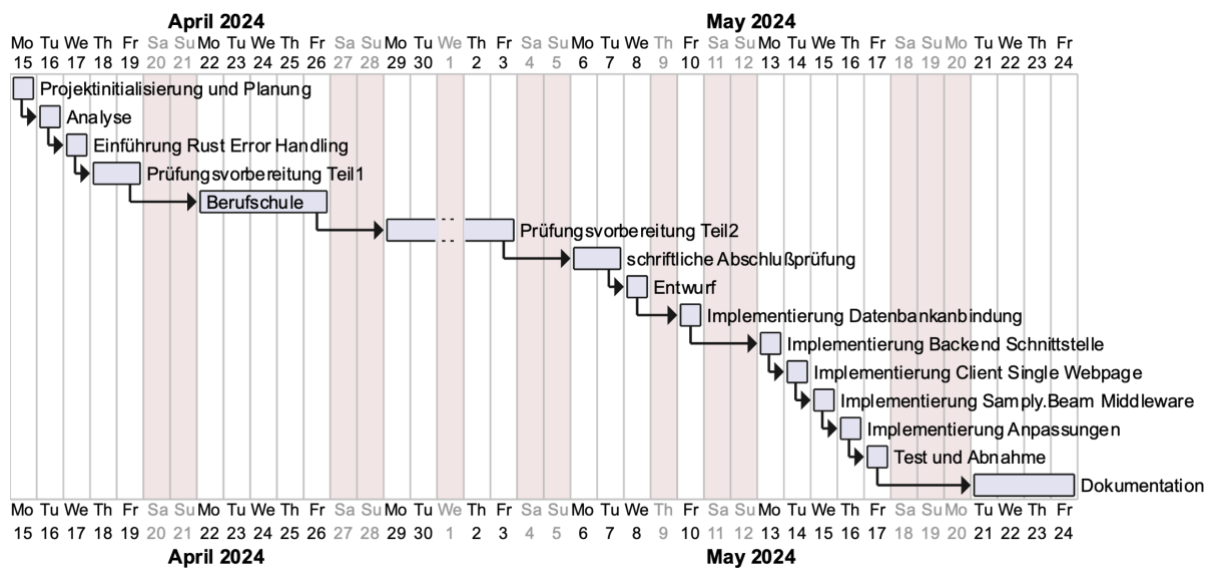


Abbildung 9: Zeitablauf und Meilensteine

## 7.11 Übersicht für Raum und Standortbelegung

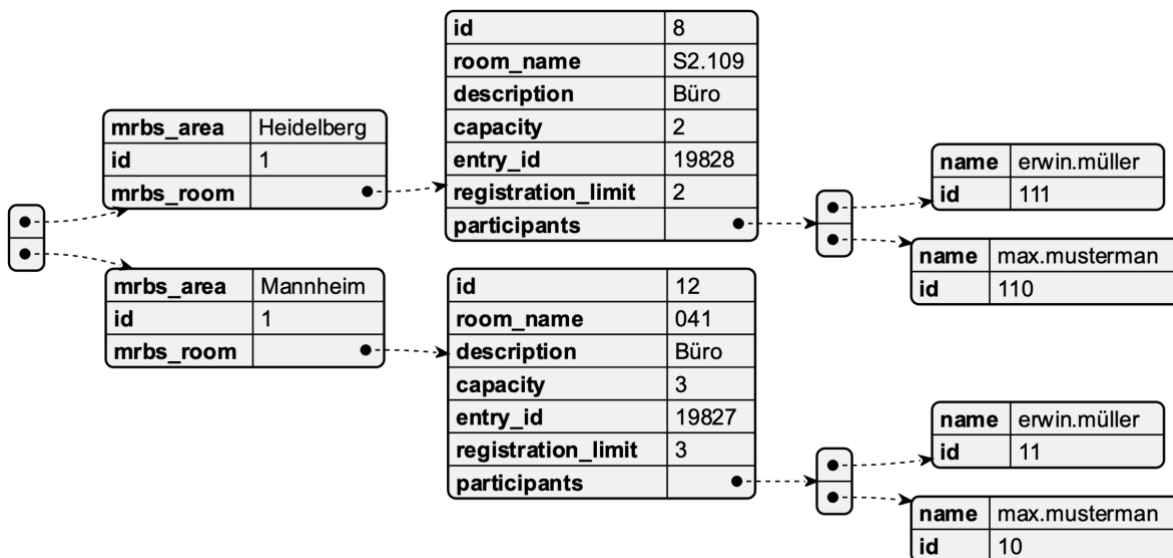


Abbildung 10: Übersicht Raum und Standortbelegung

## 7.12 Entwurf Backend mit Datenbank


|  «Module»<br>Routes  |
|---|
| <ul style="list-style-type: none"><li>○ db: DatabaseConection</li><li>○ user_name: String</li></ul>   |
| <ul style="list-style-type: none"><li>● get_overview_day(db, customday: Option&lt;String&gt;)</li><li>● postjoinroom(db, entry_id: i32, user_name)</li><li>● post_joinrooms(db, room_id: i32, days: Vec&lt;String&gt;, user_name)</li><li>● delete_participant(db, entry_id: i32)</li></ul> |

Abbildung 11: Backend Entwurf

## 7.13 Entwurf Darstellung Client Single Webpage

```
<Navigationsleiste>
Room Booking System
<Datumsfeld>
Heidelberg
<selection>(Liste mit Räumen)
<button>join
- S.2.124
  - erwin.müller <button>delete
...
Mannheim
...
```

Abbildung 12: Darstellung Single Webpage Entwurf

## 7.14 Komponenten Anbindung über Samply.Beam

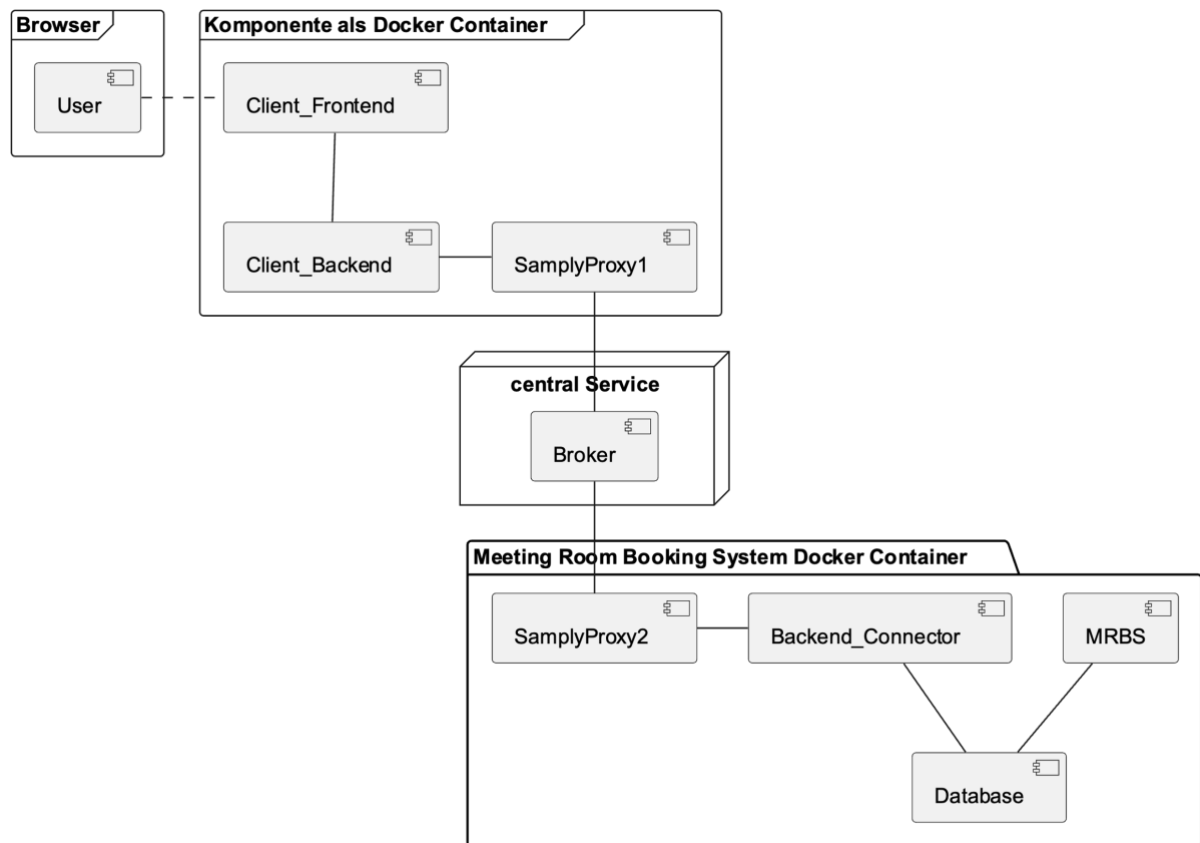


Abbildung 13: Anbindung über Samply.Beam

## 7.15 Datenstruktur für die Tagesübersicht

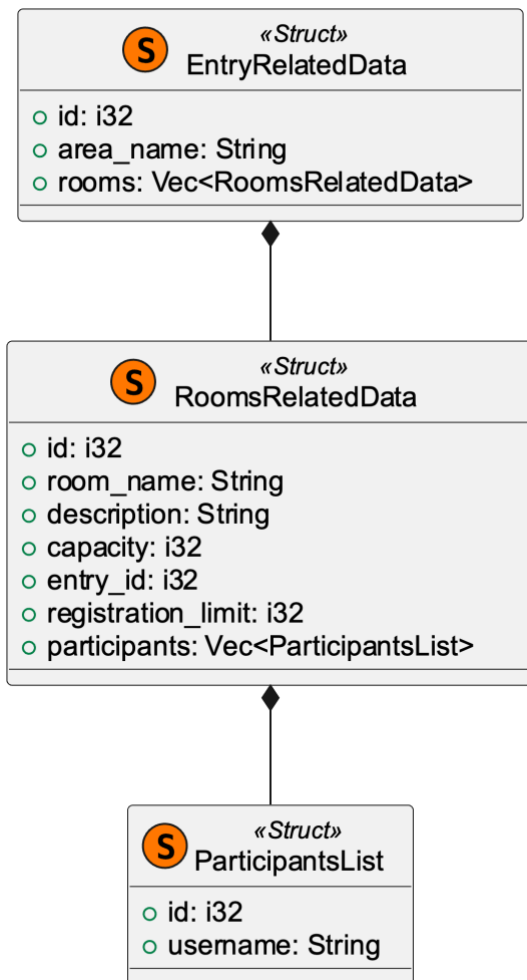


Abbildung 14: Datenstruktur für Tagesübersicht

## 7.16 Samplify.Beam Task

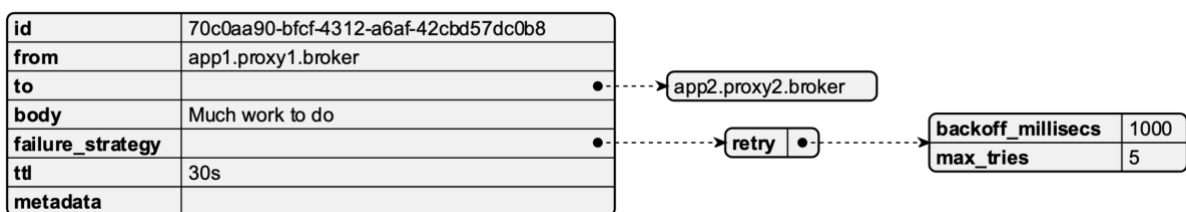


Abbildung 15: Samplify.Beam Task

## 7.17 Samplify.Beam Task Result

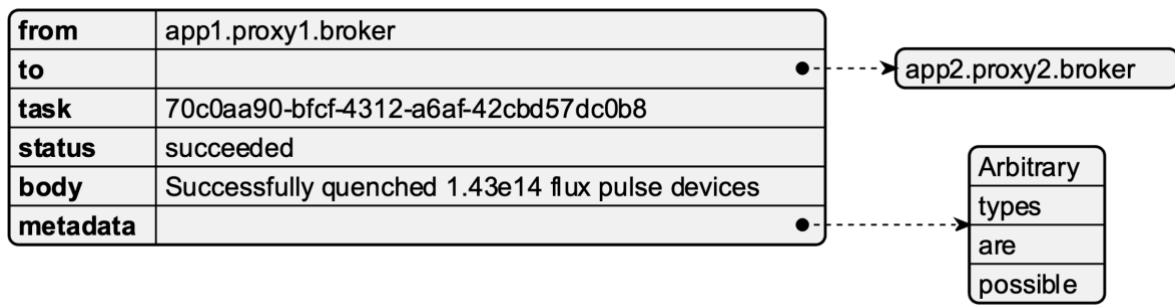


Abbildung 16: Samplify.Beam Task Result

## 7.18 Lokale Testumgebung Docker-Compose-Config

```
version: "3.1"
services:
  www:
    build:
      context: .
    ports:
      - "9000:80"
    volumes:
      - ./web:/var/www/html/
      - ./web/config.inc.php:/var/www/html/config.inc.php
    links:
      - db
    networks:
      - default
  db:
    image: mysql:8.0
    ports:
      - "3308:3306"
    command: --default-authentication-plugin=mysql_native_password
    environment:
      MYSQL_DATABASE:
      MYSQL_USER:
      MYSQL_PASSWORD:
      MYSQL_ROOT_PASSWORD:
      MYSQL_ROOT_HOST: "%"
    volumes:
      - ./backup.sql:/docker-entrypoint-initdb.d/010-tables.sql
```

```
- persistent:/var/lib/mysql
networks:
  - default
backend:
  image: mrbs_proto_backend
  # docker.verbis.dkfz.de/mrbs/backend:latest
  ports:
    - 9090:7878
  environment:
    DATABASE_URL: mysql://:@db:3306/mrbs
    BIND_ADDR: 0.0.0.0:7878
    TO_FRONTEND_PROXY: app1.proxy1.broker
    BROKERID: broker
    PROXYID: proxy2.$BROKERID
    BEAM_PROXY_URL: http://0.0.0.0:8082/
    BEAM_APP_ID: app2.$PROXYID
    BEAM_SECRET: App1Secret
    RUST_LOG: debug
  phpmyadmin:
    image: phpmyadmin
    links:
      - db:db
    ports:
      - 8888:80
  volumes:
    persistent:
```

## 7.19 Funktion für Tagesansicht Backend

```
pub async fn read_overview_day(
  db: &DatabaseConnection,
  param: Option<String>,
) -> Result<ResponseType, ErrorType> {
  let timegiven;
  // param date as rfc3339 if None today
  match param {
    Some(s) => {
      timegiven = s;
    }
    None => {
```

```

    let start_of_day = Utc::now()
    .date_naive()
    .and_hms_opt(0, 0, 0)
    .unwrap(); // only default time
    timegiven = start_of_day.and_utc().to_rfc3339();
  }
}
// get start and end of day
let result: (i32, i32) = TimeFunctions::rfc3339_to_unixtimestamp(&timegiven)
    .map_err(|_error| ErrorType::GivenTimelsNotRCF3339 )?;
// get all arears
let mut area: Vec<EntryRelatedData> = Mrbs_area::find()
    .select_only()
    .columns([mrbs_area::Column::AreaName, mrbs_area::Column::Id])
    .into_model:::<EntryRelatedData>()
    .all(db)
    .await
    .map_err(|_error| ErrorType::DBError)?;
// for all areas the related rooms
for a in &mut area {
    let area_id = a.id;
    let mut room_collection = Mrbs_room::find()
        .filter(mrbs_room::Column::Areaid.eq(area_id))
        .select_only()
        .columns([
            mrbs_room::Column::Id,
            mrbs_room::Column::RoomName,
            mrbs_room::Column::Description,
            mrbs_room::Column::Capacity,
        ])
        .into_model:::<RoomsRelatedData>()
        .all(db)
        .await
        .map_err(|_error| ErrorType::DBError)?;
    // for all rooms the entry_id and registration_limit
    for r in &mut room_collection {
        let condition_day = Condition::all()
            .add(mrbs_entry::Column::RoomId.eq(r.id))
            .add(mrbs_entry::Column::StartTime.gte(result.0))
    }
}

```



```

        .add(mrbs_entry::Column::EndTime.lte(result.1));
let entry: Option<(i32, i32)> = Mrbs_entry::find()
    .filter(condition_day)
    .select_only()
    .columns([mrbs_entry::Column::Id, mrbs_entry::Column::RegistrantLimit])
    .into_tuple()
    .one(db)
    .await
    .map_err(|_error| ErrorType::DBError)?;
// all participants for all rooms and there paricipants_id default 0
match entry {
    Some(e) => {
        let participants_collection = Mrbs_Participants::find()
            .filter(mrbs_participants::Column::EntryId.eq(e.0))
            .select_only()
            .columns([mrbs_participants::Column::Id ,mrbs_participants::Column::Username])
            .into_model::<ParticipantsList>()
            .all(db)
            .await
            .map_err(|_error| ErrorType::DBError)?;

        r.participants = participants_collection;
        r.entry_id = e.0;
        r.registration_limit = e.1;
    }
    None => {
        r.entry_id = 0;
        r.registration_limit = 0;
    }
}
// insert rooms in the specific area
a.rooms = room_collection;
}
// result Datastruct with all areas, rooms and participants
Ok(ResponseType::ResultOverviewday { result: area })
}

```

## 7.20 Funktion Arbeitsplatz buchen

```
pub async fn delete_participant(
  db: &DatabaseConnection,
  participant_id: Option<i32>,
) -> Result<ResponseType, ErrorType> {
  let id;
  // return success or error on delete
  match participant_id {
    Some(i) => id = i,
    None => return Err(ErrorType::NoEntryIDFound)?,
  }
  // delete participant in table
  Mrbs_Participants::delete_by_id(id)
    .exec(db)
    .await
    .map_err(|_error| ErrorType::FailedDelete)?;
  return Ok(ResponseType::SuccessfulDelete);
}
```

## 7.21 Funktion Arbeitsplatzbuchung löschen

```
pub async fn delete_participant(
  db: &DatabaseConnection,
  participant_id: Option<i32>,
) -> Result<ResponseType, ErrorType> {
  let id;
  // return success or error on delete
  match participant_id {
    Some(i) => id = i,
    None => return Err(ErrorType::NoEntryIDFound)?,
  }
  // delete participant in table
  Mrbs_Participants::delete_by_id(id)
    .exec(db)
    .await
    .map_err(|_error| ErrorType::FailedDelete)?;
  return Ok(ResponseType::SuccessfulDelete);
}
```

## 7.22 Funktionen von BeamBackendConnector

```
pub struct BeamBackendConnector {}

impl BeamBackendConnector {
  pub async fn process_task(db: &DatabaseConnection) {
    // listen on tasks
    let task = BEAM_CLIENT
      .poll_pending_tasks::<PlannerParams>(&BlockingOptions::from_count(1))
      .await
      .ok()
      .and_then(|mut r| r.pop());
    let Some(task) = task else {
      return;
    };
    // Is expected as "name.lastname"
    let def_name;
    if task.from.app_name().contains("-") {
      def_name = task.from.app_name().replace("-", ".");
    } else {
      def_name = task.from.app_name().to_owned();
    }
    // Match the Endpoints defined in sharetypes
    let taskresponse = match task.body.endpoint {
      Endpoints::Overviewday => read_overview_day(db, task.body.customday.clone()).await,
      Endpoints::Joinroom => joinroom(db, task.body.id, &def_name).await,
      Endpoints::Joinrooms => {
        joinrooms(db, task.body.id, task.body.days.clone(), &def_name).await
      }
      Endpoints::Deletejoin => delete_participant(db, task.body.id).await,
      Endpoints::Room => show_room(db, task.body.id).await,
    };
    // variables for TaskResult
    let for_task_id = task.id;
    let from = task.to.clone().pop().unwrap();
    let to = vec![task.from];
    let id = task.id;
    // task result
    let taskresult = TaskResult {
```

```
        from,
        to,
        task: id,
        status: beam_lib::WorkStatus::Succeeded,
        body: taskresponse,
        metadata: serde_json::Value::Null,
    };
    // send a task response back
    BEAM_CLIENT
        .put_result(&taskresult, &for_task_id)
        .await
        .unwrap();
    }
}
```

## 7.23 Template angepasste Routen für den Client Back- und Frontend

```
// *****
// FRONT END
// *****
// Front end to server svelte build bundle, css and index.html from public folder
pub fn front_public_route() -> Router {
    Router::new()
        .fallback_service(
            ServeDir::new(FRONT_PUBLIC).not_found_service(handle_error.into_service()),
        )
        .layer(TraceLayer::new_for_http())
}
// *****
// BACK END
// *****
// Back end server built form various routes that are either public, require auth, or secure login
pub fn backend(
    shared_state: Arc<store::Store>,
) -> Router {
    Router::new()
        .merge(back_token_route(shared_state))
}

// Routes for the Client-Frontend mitAPI token
```

```
pub fn back_token_route<S>(state: Arc<Store>) -> Router<S> {
  Router::new()
    .route("/overviewday", get(overviewday))
    .route("/joinroom/:id", post(BeamRequestTask::joinroom))
    .route("/joinrooms/:id", post(BeamRequestTask::joinrooms))
    .route("/deletejoin/:id", delete(BeamRequestTask::delete_participant))
    .route("/room/:id", get(BeamRequestTask::show_room))
    .route("/user", get(BeamRequestTask::get_user))
    .route_layer(middleware::from_fn_with_state(
      state.clone(),
      middlewares::auth,
    ))
    .with_state(state)
}
```

## 7.24 Funktionen des Clients um mit Beam zu kommunizieren

```
pub struct BeamRequestTask {}
impl BeamRequestTask {
  pub async fn read_overview_day(
    query_day: GetTimestamp,
  ) -> Result<Json<Vec<EntryRelatedData>>, Json<ErrorType>>> {
    // for the body a struct for all params
    let uri = Endpoints::Overviewday;
    let param = PlannerParams::new(uri, None, query_day.day.to_owned(), None);
    // variables for Beam Task
    let task_id: beam_lib::MsgId = MsgId::new();
    // create a task with the config set in main
    let task = create_beam_task(task_id, param);
    BEAM_CLIENT
      .post_task(&task)
      .await
      .map_err(|_error| Json(ErrorType::BeamNoTaskCouldBeCreated))?;
    // send task with the task_id can be listen on until there is a Beam Result Task send
    // in the body there is a ResponseType enum
    let res = handle_listen_to_beam_tasks::<Result<ResponseType, ErrorType>>>(task_id)
      .await
      .map_err(|_error| Json(ErrorType::NoBeamResponse))?
      .body;
    // the ResponseType has to be enum ResultOverview to be successfull
  }
```

```

match res {
  Ok(r) => match r {
    ResponseType::ResultOverviewday { result } => Ok(Json(result)),
    _ => Err(Json(ErrorType::NoBeamResponse)),
  },
  Err(e) => Err(Json(e)),
}
}

pub async fn joinroom(Path(id): Path<i32>) -> Result<Json<ResponseType>, Json<ErrorType>> {
  // for the body a struct for all params
  let uri = Endpoints::Joinroom;
  let param = PlannerParams::new(uri, Some(id), None, None);
  // variables for Beam Task
  let task_id: beam_lib::MsgId = MsgId::new();
  let task = create_beam_task(task_id, param);
  BEAM_CLIENT
    .post_task(&task)
    .await
    .map_err(|_error| Json(ErrorType::BeamNoTaskCouldBeCreated))?;

  let res = handle_listen_to_beam_tasks::<Result<ResponseType, ErrorType>>(task_id)
    .await
    .map_err(|_error| Json(ErrorType::NoBeamResult))?
    .body;
  match res {
    Ok(r) => Ok(Json(r)),
    Err(e) => Err(Json(e)),
  }
}

pub async fn joinrooms(
  Path(entry_id): Path<i32>,
  Json(payload): Json<Vec<String>>,
) -> Result<Json<ResponseType>, Json<ErrorType>> {
  // for the body a struct for all params
  let uri = Endpoints::Joinrooms;
  tracing::debug!("{}", payload);
  let param = PlannerParams::new(uri, Some(entry_id), None, Some(payload));

```

```

let task_id: beam_lib::MsgId = MsgId::new();
let task = create_beam_task(task_id, param);
    BEAM_CLIENT
    .post_task(&task)
    .await
    .map_err(|_error| Json(ErrorType::BeamNoTaskCouldBeCreated))?;

let res = handle_listen_to_beam_tasks::<Result<ResponseType, ErrorType>>>(task_id)
    .await
    .map_err(|_error| Json(ErrorType::NoBeamResult))?
    .body;
match res {
    Ok(r) => Ok(Json(r)),
    Err(e) => return Err(Json(e)),
}
}

pub async fn delete_participant(
    Path(entry_id): Path<i32>,
) -> Result<Json<ResponseType>, Json<ErrorType>> {
    // for the body a struct for all params
    let uri = Endpoints::Deletejoin;
    let param = PlannerParams::new(uri, Some(entry_id), None, None);
    let task_id: beam_lib::MsgId = MsgId::new();
    let task = create_beam_task(task_id, param);
    BEAM_CLIENT
    .post_task(&task)
    .await
    .map_err(|_error| Json(ErrorType::BeamNoTaskCouldBeCreated))?;
let res = handle_listen_to_beam_tasks::<Result<ResponseType, ErrorType>>>(task_id)
    .await
    .map_err(|_error| Json(ErrorType::NoBeamResult))?
    .body;
match res {
    Ok(r) => Ok(Json(r)),
    Err(e) => return Err(Json(e)),
}
}
}

```

```

pub async fn show_room(
    Path(entry_id): Path<i32>,
) -> Result<Json<RoomsRelatedData>, Json<ErrorType>> {
    // for the body a struct for all params
    let uri = Endpoints::Room;
    let param = PlannerParams::new(uri, Some(entry_id), None, None);
    // create Beam Task
    let task_id: beam_lib::MsgId = MsgId::new();
    let task = create_beam_task(task_id, param);
    BEAM_CLIENT
        .post_task(&task)
        .await
        .map_err(|_error| Json(ErrorType::BeamNoTaskCouldBeCreated))?;

    let res = handle_listen_to_beam_tasks::<Result<ResponseType, ErrorType>>>(task_id)
        .await
        .map_err(|_error| Json(ErrorType::NoBeamResult))?
        .body;

    match res {
        Ok(r) => match r {
            ResponseType::ResultRoom { result } => Ok(Json(result)),
            _ => Err(Json(ErrorType::NoBeamResponse)),
        },
        Err(e) => Err(Json(e)),
    }
}

pub async fn get_user() -> Json<std::string::String> {
    let to = env::var("BEAM_APP_ID").unwrap_or("no User".to_owned());
    let result = to.split_once(".").unwrap().0;
    Json(result.to_string())
}

```

## 7.25 Eigene Bibliothek für geteilte Datentypen

```

#[derive(Deserialize, Serialize, Debug)]
#[cfg_attr(feature = "backend", derive(sea_orm::FromQueryResult, utoipa::ToSchema, utoipa::ToResponse))]
#[cfg_attr(feature = "backend", sea_orm(entity = "Mrbs_area"))]

```



```
pub struct EntryRelatedData {
    pub id: i32,
    pub area_name: String,
    #[cfg_attr(feature = "backend", sea_orm(skip))]
    pub rooms: Vec<RoomsRelatedData>,
}

#[derive(Deserialize, Serialize, Debug)]
#[cfg_attr(feature = "backend", derive(sea_orm::FromQueryResult, utoipa::ToSchema, utoipa::ToResponse))]
#[cfg_attr(feature = "backend", sea_orm(entity = "Mrbs_room"))]
pub struct RoomsRelatedData {
    pub id: i32,
    pub room_name: String,
    pub description: String,
    pub capacity: i32,
    #[cfg_attr(feature = "backend", sea_orm(skip))]
    pub entry_id: i32,
    #[cfg_attr(feature = "backend", sea_orm(skip))]
    pub registration_limit: i32,
    #[cfg_attr(feature = "backend", sea_orm(skip))]
    pub participants: Vec<ParticipantsList>,
}

// Mrbs_Participants
#[derive(Deserialize, Serialize, Debug)]
#[cfg_attr(feature = "backend", derive(sea_orm::FromQueryResult))]
#[cfg_attr(feature = "backend", sea_orm(entity = "Mrbs_Participants"))]
pub struct ParticipantsList {
    pub id: i32,
    pub username: String
}

// timestamp as String 2023-02-02T00:00:00Z in rfc3339 for Query param
#[derive(Deserialize, Serialize)]
pub struct GetTimestamp {
    pub day: Option<String>,
}

// data to send with beam task as body
```

```
#[derive(Debug, Deserialize, Serialize)]
pub struct PlannerParams {
    pub endpoint: Endpoints,
    pub id: Option<i32>,
    pub customday: Option<String>,
    // join for many days
    pub days: Option<Vec<String>>
}

impl PlannerParams {
    pub fn new(endpoint: Endpoints, id: Option<i32>, customday: Option<String>, days: Option<Vec<String>>)
-> PlannerParams{
        PlannerParams{
            endpoint,
            id,
            customday,
            days
        }
    }
}

// valide Routes or functions for impl on the database
#[derive(Debug, Deserialize, Serialize)]
pub enum Endpoints{
    Overviewday,
    Joinroom,
    Joinrooms,
    Deletejoin,
    Room
}

// Type for successfull Response with data
#[derive(Deserialize, Serialize, Debug)]
pub enum ResponseType{
    SuccessfulJoined,
    SuccessfulDelete,
    ResultOverviewday{ result: Vec<EntryRelatedData> },
    ResultRoom{ result: RoomsRelatedData }
}

// error types can be set in the project
```

```
#[derive(Deserialize, Serialize, Debug)]
pub enum ErrorType {
    FailedDelete,
    DBError,
    RoomIsFull,
    NoRoomLimitProvided,
    NoRoomWithThisID,
    GivenTimesNotRCF3339,
    NoEntryForThisRoom,
    NoEntryIDFound,
    NoRoomIDFound,
    NoDateFound,
    NoEntry{ message: String},
    FailedJoinOn{ date: String},
    NoBeamResult,
    BeamNoTaskCouldBeCreated,
    NoBeamResponse,
    BeamNotReachable,
    BeamError,
}
```

## 7.26 JavaScript Funktionen um mit dem Client Backend zu kommunizieren

```
import { apikey, responseMrbs, user } from "./store";
import dayjs, { Dayjs } from "dayjs";

export async function getUser() {
    let res = await fetch('/user', {
        headers: {
            'Authorization': 'Bearer '+ apikey,
            'Accept': "application/json",
        },
    });
    const json_res = await res.json();
    if(res.ok){
        user.set(json_res)
    } else {
        return json_res;
    }
}
```

```
export async function getOverviewday() {
  let res = await fetch('/overviewday', {
    headers: {
      'Authorization': 'Bearer ' + apikey,
      Accept: "application/json",
    },
  });
  return await res.json();
}

export async function getCustomday(inputdate) {
  let rfc3339Date = new Date(inputdate).toISOString();
  let res = await fetch('/overviewday?day=' + encodeURIComponent(rfc3339Date), {
    headers: {
      'Authorization': 'Bearer ' + apikey,
      Accept: "application/json",
    },
  });
  const json_res = await res.json();
  if(res.ok){
    responseMrbs.set(json_res)
  } else {
    return json_res;
  }
}

export async function deleteJoin(id) {
  let res = await fetch('/deletejoin/' + id, {
    method: 'DELETE',
    headers: {
      'Authorization': 'Bearer ' + apikey,
      Accept: "application/json",
    },
  });
  console.log(res);
}

export async function setJoin(id) {
```

```
let res = await fetch('/joinroom/' + id, {
  method: 'POST',
  headers: {
    'Authorization': 'Bearer ' + apikey,
    Accept: "application/json",
  },
});
console.log(res);
}

export async function setJoins(id, weekdays, repeat_weeks) {
  let isoVec = [];

  for(let i = 7; i < (7*repeat_weeks) + 1; i += 7 ) {
    for (let val of weekdays) {
      console.log(val);
      isoVec.push( daysjs().day(val + i).toISOString() );
    }
  }

  let d = {days: isoVec};
  console.log(isoVec);
  let res = await fetch('/joinrooms/' + id, {
    method: 'POST',
    headers: {
      'Authorization': 'Bearer ' + apikey,
      Accept: "application/json",
      'Content-Type': "application/json"
    },
    body: JSON.stringify(isoVec),
  });
  console.log(res);
}
```

## 7.27 Svelte Singlepage mit Tagesansicht

```
<script>
import { onMount } from "svelte";
```

```

import { getOverviewday, deleteJoin, getUser, getCustomday, setJoin, setJoins } from "../js/fetch.js";
import { responseMrbs, user, wookingDays } from "../js/store.js";
import { DateInput } from "date-picker-svelte";

let date = new Date;
let select_entryid;
let bookingdays = [];
let booking_entryid;
let repeat_weeks = 1;
const costumdate = () => { getCustomday(date) };

onMount(() => {
  getUser();
  getOverviewday()
    .then((response) => {
      responseMrbs.set(response);
      console.log(response);
    })
    .catch((error) => {
      console.log("Error:", error);
    });
});
</script>

<h1>Raumplaner</h1>
<div>
  <container class="wider mobile">
    <h1>Welcome {user} to the room planer</h1>
    <DateInput on:select={costumdate} bind:value={date} />
    <p>{date}</p>

    {#each $responseMrbs as res}
      <h2>{res.area_name}</h2>
      <select name="allrooms" id="allrooms" bind:value={select_entryid}>
        {#each res.rooms as room}
          <option value={room.entry_id}>{room.room_name}</option>
        {/each}
      </select>
      <button type="submit" on:click={() => { setJoin( select_entryid) }}>Join Room</button>
    {/each}
  </container>
</div>

```

```

<ul>
  {#each res.rooms as room}
    <li>{room.room_name}:
      {#each room.participants as p }
        { p.username }
        <button on:click={() => { deleteJoin(p.id) }}>Delete</button> -
      {/each}
    </li>
  {/each}
</ul>

<h2>Join for next week or month</h2>
<label for="allrooms">Rooms</label>
<select name="allrooms" id="allrooms" bind:value={booking_entryid}>
  {#each res.rooms as room}
    <option value={room.id}>{room.room_name}</option>
  {/each}
</select>
<div class="week">
  {#each wookingDays as { name, num }}
    <label for={name}>{name}</label>
    <input type="checkbox" name={name} id={name} value={num} bind:group={bookingdays} >
  {/each}
</div>
<br>
<label for="weekpicker">Pick the amount of week to repeat</label>
<input type="range" name="weekpicker" min="1" max="15" bind:value={repeat_weeks}>
<button type="submit" on:click={() => { setJoins(booking_entryid , bookingdays, repeat_weeks) }}>Join
Room {bookingdays}</button>
  {/each}
</container>
</div>

```

## 7.28 Angepasste Funktionen von Samply.Beam

```

pub fn create_beam_task(
  id: beam_lib::MsgId,
  query: PlannerParams,
) -> TaskRequest<PlannerParams> {

```

```

let to = vec![AppId::new_unchecked(env::var("TO_API_PROXY").unwrap().to_owned());]
let from = AppId::new_unchecked(env::var("BEAM_APP_ID").unwrap().to_owned());
let metadata = if let Some(project) = &CONFIG.project {
    serde_json::json!({
        "project": project
    })
} else {
    serde_json::Value::Null
};

TaskRequest {
    id,
    from, // : CONFIG.beam_app_id.clone(),
    to,
    metadata,
    body: query,
    failure_strategy: beam_lib::FailureStrategy::Retry {
        backoff_millisecs: 1000,
        max_tries: 5,
    },
    ttl: "360s".to_string(),
}

}

#[derive(Deserialize)]
struct ListenQueryParameters {
    wait_count: u16,
}

pub async fn handle_listen_to_beam_tasks<T: DeserializeOwned + 'static>(
    task_id: MsgId,
) -> Result<TaskResult<T>, ErrorType> {
    let results = BEAM_CLIENT
        .poll_results(&task_id, &BlockingOptions::from_count(1))
        .await;
    results
        .map_err(|_error| ErrorType::NoBeamResult)?
        .pop()

```



```
.ok_or(ErrorType::NoBeamResult)
}
```

## 7.29 UML BackendConector und Client

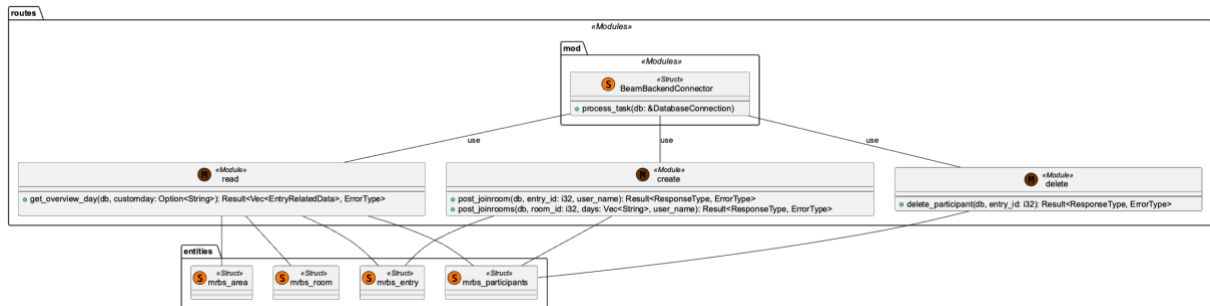


Abbildung 17: BackendConnector UML

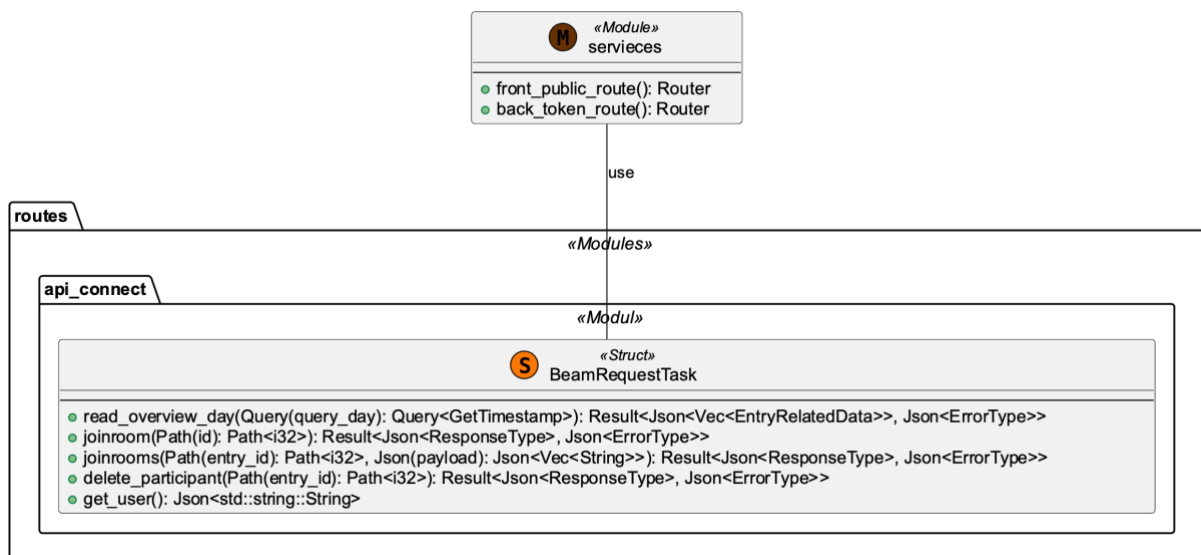


Abbildung 18: Client UML

## 7.30 SharedTypes Bibliothek

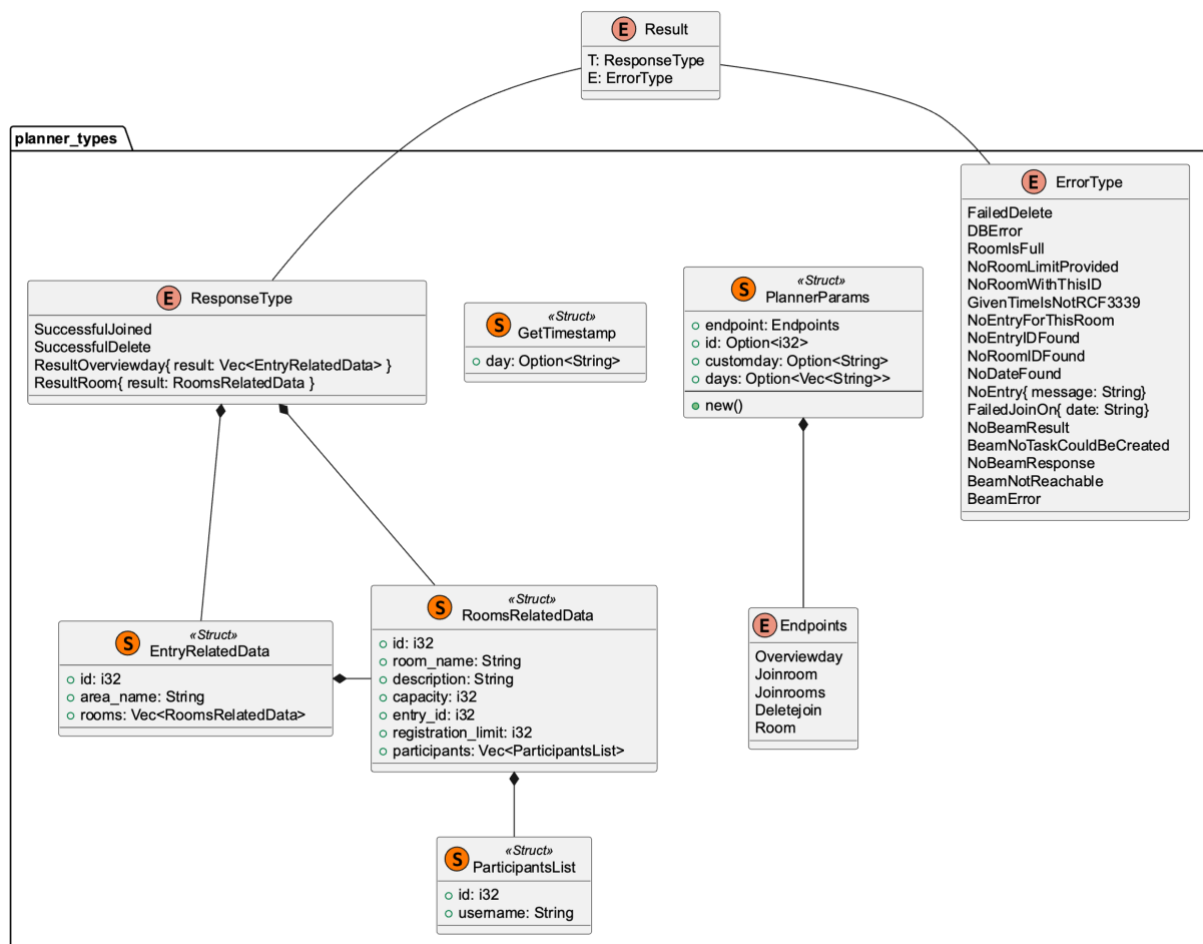


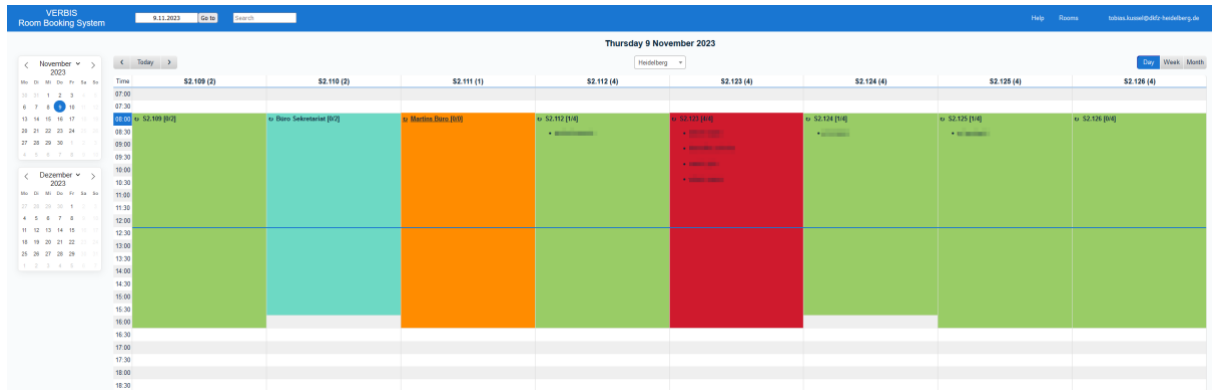
Abbildung 19: SharedTypes Bibliothek

## 7.31 Projektauftrag

# Projektbeschreibung „Raumbuchungssystem über Samply.Beam“

### Einleitung

Da die Mitarbeiter\*innen der Abteilung sowohl in den Büroräumen des DKFZ in Heidelberg, als auch in den Räumlichkeiten der UMM Mannheim tätig sind, sind alle Arbeitsplätze für das flexible Arbeiten beliebiger Mitarbeiter\*innen ausgelegt. Um die Team-Absprachen zu vereinfachen und einen Überblick über die genutzten Ressourcen zu erlangen buchen sich die Mitarbeiter\*innen über ein elektronisches Raumbuchungssystem in die gewünschten Arbeitsplätze ein.



Das Raumbuchungssystem ist aus Gründen des Datenschutzes nur aus dem internen Netzwerk des DKFZ erreichbar, was die Nutzung zur Planung der Anwesenheit erschwert. Zum morgendlichen überprüfen, ob z.B. alle Teilnehmer einer Projektgruppe für den Tag in Heidelberg oder Mannheim eingebucht sind, muss erst das Arbeitslaptop gestartet, die VPN-Verbindung aufgebaut, und schließlich der Raumplaner geprüft werden.

Eine Möglichkeit zur „Verschlankung“ der Nutzung des Systems stellt eine Kommunikation mit dem Raumbuchungssystem über die Kommunikationsmiddleware „Samply Beam“

(<https://github.com/samply/beam>) da. Samply.Beam wurde als Message Broker Lösung für sensibelste Netzwerkzonen, z.B. Kliniknetzwerke konzipiert und könnte die Datenschutzkonforme Kommunikation für autorisierte Nutzer\*innen erlauben.

Ziel dieses Projekts ist die Nutzbarmachung des Raumbuchungssystems mit einer nutzerfreundlichen, Beam-Basierten Softwarelösung.

### Aktueller Stand

- Raumbuchungssystem nutzt PHP
- Vorarbeit mit Java API
- Vorarbeit mit Rust API
  - Prototypisches Frontend

### Notwendige Schritte

- Prüfen der Vorarbeiten
- Aufsetzen Beam Broker
- Handling Authentication
- Switch auf Processing echter Daten
- Definition Beam API
- Frontend anpassen
- Integration Beam-Proxy und Frontend (Android App?)
- CI/CD für Nutzerspezifische Pakete

## 7.32 Zeitplanung

| Reihenfolge | Projektphasen  | Zeit in h |
|-------------|--|-----------|
| 1           | Projekt initialisierung  | 1         |
| 1.1         | Kundengespräch   | 1         |
| 2           | Planung  | 3         |
| 2.1         | Erarbeitung der Anforderungen  | 1         |
| 2.2         | Prüfen der Vorarbeiten   | 1         |
| 2.3         | Erstellen eines Zeitplans  | 1         |
| 2.4         | Abgleichen der Anforderungen mit Auftraggeber                                    | 1         |
| 3           | Analyse  | 8         |
| 3.1         | Ist-Analyse  | 1         |
| 3.2         | Soll-Analyse   | 1         |
| 3.3         | Konzepte entwickeln  | 4         |
| 3.4         | Entscheidung für Projektumsetzung und Kundenabsprache                            | 2         |
| 4           | Entwurf  | 7         |
| 4.1         | Komponenten Diagramm des Umsetzungskonzepts                                      | 1         |
| 4.2         | Datenbank Anbindungsentwurf  | 2         |
| 4.2         | Nutzwertanalyse zur Auswahl der Frameworks                                       | 1         |
| 4.3         | Minimales Frontend entwerfen   | 0.5       |
| 4.4         | REST API Ressourcen und Schnittstellen entwerfen                                 | 1.5       |
| 4.5         | Deploymentdiagramm erstellen   | 1         |
| 5           | Implementierung  | 37        |
| 5.1         | Lokale Entwicklungsumgebung mit Docker Containern Konfigurieren als Testumgebung | 1         |
| 5.2         | Projekt Struktur für Routen und Datenbankanbindung und Zugriff                   | 8         |
| 5.3         | REST API umsetzen  | 8         |
| 5.4         | Frontend Single Page umsetzen  | 7         |
| 5.5         | BEAM als Middleware einrichten   | 8         |
| 5.6         | Docker Container für Live System anpassen  | 1         |
| 5.7         | Erstellen einer Pipeline GitLab für individuelles Zertifikat (Endprodukt Client) | 4         |
| 6           | Test   | 3         |
| 6.1         | Code Review  | 2         |
| 6.2         | Funktionsstest   | 1         |
| 7           | Abnahme und Produktivsetzung   | 2         |
| 7.1         | Übergabe und Unterstützung bei Anbindung der Komponenten                         | 1         |
| 7.2         | Testen des individuellen Clients und Kunden Abnahme                              | 1         |
| 8           | Dokumentation  | 15        |
| 8.1         | Erstellen einer Dokumentation  | 11        |
| 8.2         | GitLab Dokumentation und Installationsanweisung                                  | 2         |
| 8.3         | Kundendokumentation REST Schnittstellen  | 2         |
| 9           | Sonstiges  | 4         |
|             | Puffer   | 4         |
|             | Gesamt   | 80        |

## 8 Kundendokumentation

### 8.1 Room Booking System over Samply.Beam

#### 8.1.1 Description "Raumbelegungssoftware"

This is the new Roombooking System over Samply.Beam, so you don't have to login via VPN. The GUI gives you a simple day overview as list from Mannheim and Heidelberg. Access it under localhost:9020 in default settings. In the future you can also book a workspace for the next weeks based on a weekday.

#### 8.1.2 Technologiestack

- Rust Axum als Backend
- Svelte als Frontend
- Framework: Axum, SeaORM [Template Readme](#)

#### 8.1.3 Installation

You need to clone this Repo and Docker must be running to as basic. First you need to change the following in beam-proxy:

```
PROXY_ID: firstname-lastname.test-no-real-data.broker.samply.de
```

Second in frontend:

```
BEAM_APP_ID=firstname-lastname.$PROXYID  
PROXYID=firstname-lastname.$BROKERID
```

Third in secret:

```
file: ./pem/firstname-lastname.priv.pem
```

Fourth We created a certificate enrollment companion tool, assisting the enrollment process. Please run the docker image via:

```
docker run --rm -ti -v $(pwd)/pem:/pki samply/beam-enroll:latest --output-  
file pki/firstname-lastname.priv.pem --proxy-id firstname-lastname.test-no-  
real-data.broker.samply.de
```

and follow the instructions on the screen. The tool generates the private key file in the given directory and prints the CSR to the console -- ready to be copied into an email to the central CA's administrator without the risk of accidentally sending the wrong, i.e. private, file. Fifth

Raumbelegungssoftware über Samply.Beam

```
docker-compose up
```

Sixth You can access in default settings the GUI under localhost:9020